Universidad Carlos III de Madrid
Escuela Politécnica Superior

Una Arquitectura de Subsunción para robots Pioneer P3-DX

Proyecto Fin de Carrera

Ingeniería Informática

Autor: Juan José María González
Tutores: D. Agapito I. Ledezma Espino
D. Fernando Fernández Rebollo
Año: 2009

AGRADECIMIENTOS

Quisiera aprovechar la oportunidad que se me brinda para agradecer a todas esas personas que me han apoyado tanto durante este tiempo. En primer lugar, y como no podía ser de otra forma, agradecer a mis padres y mi hermana por todo su apoyo, tanto en los momentos buenos como en los más difíciles, donde nunca me han fallado.

En segundo lugar, quisiera agradecer a Fernando y Agapito, por ofrecerme este proyecto que me permite culminar la carrera e iniciar la transición al mundo laboral, siguiente etapa de mi vida en la que podré afrontar nuevos retos. Además, como olvidarse en un momento así, de aquellos compañeros como Patricia y José, que durante todo este tiempo han compartido conmigo tantos buenos momentos, así como esfuerzos y sacrificios.

Por último, pero no por ello menos importante, a mis amigos, por su apoyo y confianza, fortaleciéndome en los momentos más difíciles. Así como a todos los profesores que me han encaminado hacia este momento.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1: **INTRODUCCIÓN**

1.1. Marco del proyecto fin de carrera

En apenas cuatro décadas, la robótica ha pasado de ser un producto de la imaginación de determinados autores de ciencia ficción, a una realidad implantada en las más dispares áreas. Desde la domótica o robótica de uso doméstico hasta las grandes cadenas de producción en las más importantes factorías, emplean aparatos mecánicos que se pueden agrupar bajo el término “robot”.

La palabra, de origen checo, robot, da idea de un trabajo forzado y fue empleada por primera vez para denominar a estos autómatas mecánicos en 1921 por Karen Capek en su obra teatral R.U.R. Más adelante Isaac Asimov la implantaría definitivamente en sus libros sobre robots.

En el campo de la industria, la integración de los grandes robots manipuladores en la cadenas de montaje de la principales factorías, ha mejorado la productividad y eficiencia de estas de un modo que apenas se podía intuir antes de la fabricación industrial de estos mecanismos; ciertos trabajos industriales como la soldadura y la pintura en las líneas de producción no se conciben ya sin un robot que realice la mayor parte o casi la totalidad de estos trabajos.

También en sectores como la medicina, debido a la gran precisión que se requiere o en trabajos con entornos agresivos o asépticos, se está empezando a implantar el uso de robots de manera generalizada.

A pesar de los avances que durante las últimas décadas se han registrado, principalmente en cadenas de montaje, en tecnología aeroespacial o en telemedicina, se trata de robots o de máquinas automatizadas muy precisas, pero diferentes al "androide" con el que sueñan los expertos en robótica. La robótica, aún se encuentra muy lejos de las fantasías que plantea George Lucas en su saga cinematográfica "StarWars" y robots como C3PO y R2-D2, con una autonomía total de movimientos, una percepción completa de su entorno y un razonamiento casi humano.

El desarrollo de un sistema mecánico que pueda desplazarse con total autonomía en cualquier entorno basándose en lo que percibe de este a través de sus sensores, realizando a su vez tareas que se pueden considerar útiles, no está todo lo avanzado que se podría suponer. Actualmente los robots que operan en entornos variables, están en su mayor parte teleoperados por un operador humano, que es el que aporta la percepción y capacidad de decisión al sistema. Por otro lado, la gran mayoría de los robots autónomos, realizan sus funciones en entornos controlados, es decir, entornos en los que se conocen de antemano y con total precisión tanto las formas y dimensiones como los obstáculos que puedan existir en el mismo.

Cuando se trabaja con un robot móvil en un entorno controlado, el problema se reduce en un grado muy elevado, ya que el robot puede tener programados de antemano los movimientos que ha de realizar o calcularlos mediante sus algoritmos sin una gran dificultad. Los robots móviles autónomos, forman una familia de máquinas, que se ajustan a este perfil, y su desarrollo e implantación en la industria actual, les auguran un prospero futuro y un papel muy importante en dicho futuro.

Los Robots Móviles Autónomos conforman uno de los campos en auge dentro del desarrollo de la robótica actual. El concepto de estos robots surge ante el deseo de conseguir un sistema móvil con capacidad de desarrollar actividades útiles durante un periodo de tiempo considerable sin la necesidad de la intervención humana.

Dentro de la familia robótica, los robots móviles autónomos están considerados como una subclase de los Vehículos Sin Piloto. Mientras que los vehículos sin piloto se definen como dispositivos móviles que realizan actividades sin piloto, aunque pueden estar manejados por control remoto, los robots móviles autónomos, en cambio, no necesitan ningún tipo de participación humana en su navegación.

Este grado de *autonomía* de los robots móviles autónomos, les obliga a poseer cierto tipo de inteligencia que les permita, a partir de las órdenes de alto nivel recibidas del operador humano, realizar algunas de sus tareas sin la intervención de éste, seleccionando el robot los criterios a seguir para la realización de las mismas.

Los robots tienen numerosos campos de aplicación dentro de la industria actual, entre las cuales se pueden destacar: sistemas de producción industrial, sistemas de seguridad o sistemas de mantenimiento.

1.2. Objetivos del proyecto fin de carrera

El presente documento muestra el trabajo desarrollado en el proyecto fin de carrera "Una Arquitectura de Subsunción para Robots Pioneer P3-DX".

Para el desarrollo del proyecto, se ha trabajado con robots pertenecientes a la familia de robots de movimiento de la casa Pioneer, en concreto con el modelo P3-DX. Estos robots se basan en movimientos mediante tracción diferencial, es decir que poseen dos ruedas motrices, y una tercera libre para mantener la estabilidad. Este sistema de tracción diferencial hace que el robot presente una alta movilidad y un gran grado de maniobrabilidad en sus movimientos.

El objetivo es conseguir la navegación del robot, es decir, que sea capaz de moverse por diferentes entornos, reaccionando y recuperándose de situaciones inesperadas. Para conseguir esto, se implementa una arquitectura de subsunción, la cual se estructura en capas donde cada una de ellas presenta una capacidad independiente, dotando de una funcionalidad simple al robot.

Las funcionalidades que aportan los distintos niveles al robot, pueden clasificase en dos tipos:

- **Básicas**: funcionalidades que aseguran el funcionamiento y recuperación del robot ante situaciones indeseadas.

- **De alto nivel**: funcionalidades que dotan al robot de una utilidad más específica.

Adicionalmente, se ha marcado como objetivo el aplicar técnicas de minería de datos y aprendizaje automático para el desarrollo de funcionalidades del robot.

A lo largo del presente documento se describirá en detalle todo el trabajo realizado en el proyecto fin de carrera: arquitectura, descripción de niveles y comportamientos, problemas encontrados, decisiones tomadas, etc.

1.3. Estructura del documento

2. Estado de la cuestión: Este capítulo ofrece una visión general de los diferentes aspectos tenidos en cuenta durante el desarrollo del proyecto. Todo ello conlleva una descripción de los robots Pioneer P3-DX, los cuales se utilizaron para el desarrollo del proyecto. Del mismo modo se describirá el simulador utilizado para la realización de pruebas. Por otro lado, se presentará la arquitectura de subsunción que se pretende desarrollar.

Otro aspecto a describir es la técnica de minería de datos utilizada para la calibración de los sensores para el buen funcionamiento de los comportamientos desarrollados.

3. Objetivos del PFC: En este capítulo se centran y describen los objetivos relativos al diseño y construcción de la arquitectura de subsunción y funcionalidades relacionadas.

4. Trabajo Realizado: En este apartado se relata todo el trabajo llevado a cabo en el desarrollo del presente proyecto fin de carrera. Adicionalmente se incluirán los resultados obtenidos en el desarrollo, pero no solo los resultados directamente ligados al objetivo principal del proyecto "*Una Arquitectura de Subsunción para robots Pioneer P3-DX*", sino también resultados generales y productos obtenidos.

5. Conclusiones y Trabajos Futuros: En este capítulo se detallan las conclusiones principales sobre el desarrollo realizado, así como algunas consideraciones personales acerca del robot y sus capacidades. Se indicarán posibles líneas futuras de trabajo a aplicar a estos robots de Pioneer.

CAPÍTULO 2: **ESTADO DE LA CUESTIÓN**

El propósito del presente capítulo es situar el proyecto fin de carrera realizado dentro de un conjunto más amplio de desarrollos científicos. Este encuadre es importante, ya que de modo algo simplificador, ayuda a la valoración del trabajo realizado.

2.1. Paradigmas de control

Un paradigma de control es una estructura computacional formada por diversos módulos interconectados y cuya función es garantizar el cumplimiento por parte del robot de la o las tareas asignadas en condiciones estables, seguras y en tiempos apropiados. Se pueden encontrar diferentes clasificaciones de paradigmas de control según su diseño: jerárquicos, reactivos o híbridos [25].

2.1.1. Paradigmas Jerárquicos

El paradigma jerárquico es el más antiguo. En este paradigma el proceso que se realiza es el de: percibir el mundo, planificar y actuar; tal y como se ve representado en la figura 2.1.

Figura 2.1: Representación de arquitecturas jerárquicas

La percepción de datos es recogida en un modelo global del mundo, una simple representación que el planificador puede usar y puede enviar a las acciones. La construcción genérica del modelo del mundo es muy costosa computacionalmente.

La figura 2.2 muestra al paradigma jerárquico representado como un flujo transitivo de los acontecimientos a través de las primitivas (percibir, planificar y actuar). Se aprecia como, lamentablemente, el flujo de los acontecimientos hace caso omiso de las pruebas biológicas que indican que la percepción de la información puede estar directamente unida a una acción.

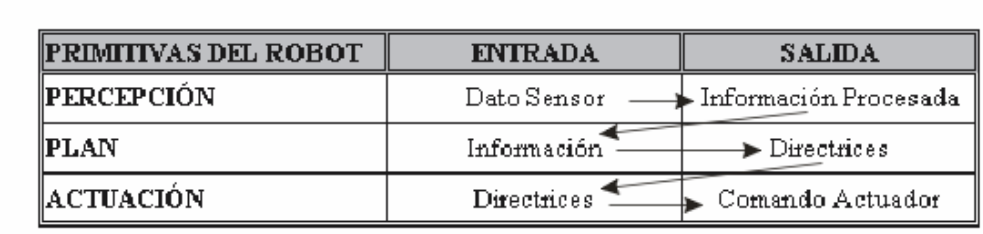

Figura 2.2: Primitivas en arquitecturas jerárquicas

La principal ventaja de este paradigma es que proporciona un orden en la relación entre percepción, planificación y acción. Por otro lado, presenta una notable desventaja, la planificación, ya que en cada ciclo de actualización, el robot tiene que actualizar un modelo del mundo y, a continuación, hacer una planificación, siendo este un proceso lento, creándose un cuello de botella en la ejecución. Otra desventaja reside en lo ya comentado, la percepción y la actuación no están en contacto, lo que conlleva que no se den situaciones de estimulo-acción tal y como se dan en la naturaleza.

2.1.2. Paradigmas Reactivos

El paradigma reactivo apareció como contrapartida al paradigma jerárquico y dio lugar a interesantes avances en robótica. Esta fue muy usada en robótica desde 1988 hasta 1992, cuando se tendió a utilizar un paradigma híbrido. Este paradigma tuvo una gran impulsión gracias a dos tendencias: la primera fue que surgió un movimiento popular entre los investigadores de IA para investigar la biología y la psicología cognitiva con el fin de examinar la vida como ejemplo de inteligencia; la segunda fuente que impulsó el desarrollo de este paradigma, fue la rápida disminución de los costes de hardware, junto con el aumento de las capacidades de los ordenadores.

El paradigma reactivo desecha el tener que planificar antes de realizar cada acción, ya que establece una relación del tipo “percibir-actuar” tal y como se muestra en las figuras 2.3 y 2.4. Es decir, mientras que en el paradigma jerárquico se asumía que la entrada de una acción debe ser siempre el resultado de un plan, el reactivo asume que la entrada a una acción será siempre la salida del procesado de la información captada.

Figura 2.3: Representación arquitecturas reactivas

PRIMITIVAS DEL ROBOT	ENTRADA	SALIDA
PERCEPCIÓN	Dato Sensor	Información Procesada
ACTUACIÓN	Información Procesada	Comando Actuador

Figura 2.4: Primitivas en arquitecturas reactivas

El robot debe poseer múltiples instancias de "percibir-actuar", ya que en otro caso el robot tan solo podría realizar una única tarea. El acoplamiento de estas múltiples instancias es concurrente, y se denominan comportamientos, los cuales procesan los datos y computan la mejor acción a realizar, independientemente de lo que esté realizando otro comportamiento.

Una de las grandes ventajas que presentan las arquitecturas que implementan este paradigma es la modularidad, ya que las acciones y percepciones necesarias para realizar una tarea se descomponen en comportamientos. Por el contrario, como desventaja se encuentra que los sistemas reactivos se limitan a aplicaciones que se pueden lograr con comportamientos reflexivos, por lo que no puede ser transferida a ámbitos en los que el robot tiene que hacer planificación, el razonamiento acerca de la asignación de recursos, etc.

2.1.3. Paradigmas Híbridos

Los paradigmas híbridos surgieron en 1990 y continúan siendo actualmente un sector en el que se está trabajando. En un paradigma híbrido, el robot primero planifica la mejor manera de descomponer una tarea en subtareas (también llamados planificación de la misión) y, a continuación cuáles son los comportamientos adecuados para realizar cada subtarea. Una vez realizado esto, los comportamientos comienzan a ejecutarse del mismo modo que para las arquitecturas reactivas. La organización de este tipo de arquitecturas es planificar, percibir-actuar (PI, Pr-A), donde la coma indica que la planificación se realiza en un paso y, a continuación, sensores y actuadores trabajan en conjunto (figuras 2.5 y 2.6).

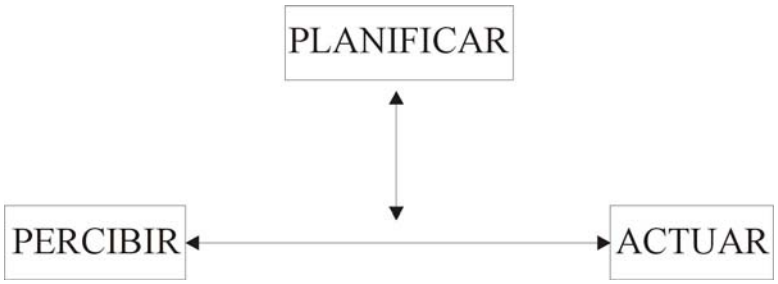

Figura 2.5: Representación de arquitecturas híbridas

PRIMITIVAS DEL ROBOT	ENTRADA	SALIDA
PLAN	Información	Directrices
PERCEPCIÓN-ACTUACIÓN	Dato Sensor	Comando Actuador

Figura 2.6: Primitivas en arquitecturas híbridas

El comportamiento de las primitivas en la organización de las arquitecturas híbridas es también una mezcla de los estilos reactivo y jerárquico. La información captada por los sensores es encaminada hacia los comportamientos que necesitan de esa información; pero dicha información captada por los sensores también está disponible para el planificador, ya que puede ser necesaria para la construcción de una tarea orientada al modelo del mundo. Por tanto el planificador puede también “escuchar” la percepción realizada por cada comportamiento.

La inclusión del componente deliberativo permite a las arquitecturas híbridas utilizarse para las situaciones en las que los sistemas puramente reactivos no eran válidos. Sin embargo, se mantiene las ventajas de las arquitecturas reactivas, ya que la división entre la reacción y la deliberación permite que la parte reactiva pueda ser usada sólo para aplicaciones puramente reactivas.

2.2. Arquitectura de Subsunción

La arquitectura de subsunción desarrollada por Rodney Brook’s es la más influyente de los paradigmas de control reactivos. Tuvo una gran influencia en la construcción de robots de ámbitos naturales, es decir de robots que simulan organismos existentes en la naturaleza; Estos robots parecían insectos del tamaño de una caja de zapatos, con seis patas y unas antenas. En muchas aplicaciones, los comportamientos están incorporados directamente en el hardware o en pequeños micro-procesadores, lo que permite al robot tener capacidad de computación incorporada. Además, los robots fueron los primeros en ser capaces de caminar, evitar colisiones y subir obstáculos sin aplicar arquitectura jerárquica.

En la arquitectura de subsunción el término "comportamiento" tiene un sentido menos preciso que en otras arquitecturas. En esta arquitectura, un comportamiento es una red de sensores y actuadores que realizan una tarea, con lo cual los comportamientos quedan algo difuminados en la arquitectura.

Por tanto, los comportamientos se pueden resumir como la interacción estímulo-respuesta, sin un programa externo que los controle y coordine explícitamente. Existen cuatro aspectos interesantes en el control y relaciones de comportamientos:

1. Los módulos se agrupan en capas de competencia. Las capas reflejan una jerarquía de la inteligencia, o de competencia. Las capas inferiores encapsulan las funciones básicas de supervivencia, tales como evitar las colisiones, mientras que niveles más altos crean acciones más dirigidas a un objetivo, por ejemplo de mapeo. Cada una de las capas puede considerarse como un comportamiento abstracto para una determinada tarea.

2. Los módulos de las capas superiores pueden anular, o subsumir, el resultado de comportamientos de la capa inmediatamente inferior. Las capas de comportamientos operan de manera simultánea e independiente, por lo que debe existir un mecanismo para manejar los posibles conflictos. La solución en los conflictos es determinar el ganador, el cual será la capa superior.

3. Se evita la utilización de estados internos (representación persistente el mundo y posicionamiento en él). Si el robot dependiese de una representación interna sería un problema debido a las posibles variaciones de la realidad.

4. Una tarea se realiza mediante la activación de la capa apropiada, la cual después activa la capa justo inferior, y así sucesivamente. Sin embargo, en la práctica, los sistemas de subsunción de este estilo no son fáciles de modificar, es decir, que no puede ser obligado a hacer otra tarea sin ser reprogramados.

Para resumir la subsunción:

- La arquitectura de subsunción presenta una floja definición de comportamiento, definiéndolo como un firme acoplamiento de sensores y actuadores.

- Las capas mas altas pueden subsumir e inhibir comportamientos a la capa inferior, pero los comportamientos en las capas inferiores no son nunca reescritos o sustituidos. Desde un punto de vista de programación, esto puede parecer extraño. Sin embargo, imita la evolución biológica.

- El diseño de capas y comportamientos que conforma una arquitectura de subsunción, como ocurre con todos los diseños de comportamientos, es difícil, es más un arte que una ciencia. Este hecho es válido para cualquier arquitectura reactiva.

- Los comportamientos son ejecutados por un estímulo del entorno.

- La subsunción resuelve el problema de tener la necesidad de modelar el mundo que rodea al robot. Asimismo, no tiene que preocuparse por los cambios del entorno y la realidad de estos, ya que los comportamientos no recuerdan el pasado, sino que los comportamientos simplemente responden a cualquier estímulo con independencia del medio ambiente en la situación anterior.

- La percepción por parte de los sensores puede ser compartida por diferentes capas.

2.3. Aprendizaje Automático

En general, el área de aprendizaje automático dentro de la inteligencia artificial se encarga de crear programas inteligentes en un sentido amplio, es decir que sea capaz de utilizando la experiencia, mejorar automáticamente su rendimiento. La idea es buscar métodos capaces de aumentar las capacidades de las aplicaciones habituales (sistemas basados en el conocimiento, tratamiento del lenguaje natural, robótica, etc.) de manera que puedan ser más flexibles y eficaces.

El área de aprendizaje automático es relativamente amplia, y ha dado lugar a muchas técnicas diferentes de aprendizaje. Dependiendo del tipo de realimentación, se puede clasificar el tipo de aprendizaje en tres grupos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado consiste en un tipo de aprendizaje automático donde al algoritmo que se utiliza se le proporcionan una serie de ejemplos, los cuales han sido clasificados con anterioridad. De esta manera, en el proceso de aprendizaje, el algoritmo compara su salida actual con la clasificación del ejemplo para luego realizar los cambios que sean necesarios.

En el caso del aprendizaje supervisado, cada ejemplo se puede expresar mediante la forma atributo-valor. Los atributos que forman parte de las instancias pueden ser categóricos o nominales y numéricos. Por ejemplo, el atributo estado civil es categórico con sus posibles valores (soltero o casado). Por otro lado, los valores de los atributos edad, o peso pueden ser numéricos y, en consecuencia, pueden ser llamados continuos.

Las variables se pueden dividir en discriminantes o dependientes; en el aprendizaje supervisado se obtiene información de la variable dependiente a partir de los valores de las discriminantes. Si los valores de la variable dependiente pertenecen a un número definido de clases, se dice que es una tarea de clasificación y si el valor es continuo la tarea es una regresión.

El conjunto de todos los posibles valores que pueden tomar los atributos discriminantes se conoce como espacio de las instancias o espacio de entrada. El conjunto de posibles valores de la variable dependiente se conoce como espacio de salida.

Generalmente, cuando se lleva a cabo una tarea de clasificación o de regresión, se utiliza un conjunto de instancias para que el algoritmo de aprendizaje construya un clasificador. Este conjunto de ejemplos es llamado conjunto de entrenamiento. Para validar este clasificador, generalmente se utiliza un conjunto de instancias que no se ha utilizado para construir el clasificador, este conjunto recibe el nombre de conjunto de prueba.

Ejemplos de una tarea de clasificación pueden ser: predecir el tiempo, determinar si un paciente puede tener una enfermedad o no, etc.

A la hora de evaluar la precisión de un clasificador se utiliza el conjunto de test sobre el cual se obtiene una precisión de clasificación que es calculada basándose en los ejemplos del conjunto de test que el clasificador ha clasificado correctamente. Al medir la precisión de un clasificador se puede utilizar también la *tasa de error* que es el complemento de la precisión de clasificación.

Dentro del aprendizaje supervisado, de acuerdo al tipo de representación de los datos de entrada, se puede hacer una clasificación en dos grupos: los representados en la forma atributo-valor y los que están representados en forma de relaciones.

Dentro del grupo de algoritmos que utilizan la representación de atributo-valor existen a su vez dos grupos: algoritmos simbólicos y subsimbólicos. Entre los algoritmos simbólicos se pueden destacar los árboles de decisión. Ejemplos de aprendizaje subsimbólico son las redes de neuronas y los algoritmos genéticos.

El aprendizaje no supervisado realiza agrupamientos en forma natural sobre los patrones de entrada. A diferencia del aprendizaje supervisado, en este tipo de aprendizaje no se conoce “a priori” el atributo dependiente.

Ejemplos de este tipo de aprendizaje son los algoritmos de agrupamiento.

En el aprendizaje por refuerzo, el algoritmo utilizado recibe las entradas y una evaluación de tal manera que el algoritmo debe aprender qué acción es la que brinda más rendimiento a largo plazo.

2.3.1. WEKA

WEKA es una herramienta de minería de datos y aprendizaje automático. Las siglas WEKA son acrónimo de *Waikato Environment for Knowledge Analysis (Entorno para Análisis de Conocimiento)*, es un entorno para experimentación de análisis de datos que permite aplicar, analizar y evaluar las técnicas más relevantes de análisis de datos, principalmente las provenientes del aprendizaje automático, sobre cualquier conjunto de datos del usuario. Para ello únicamente se requiere que los datos a analizar se almacenen con un cierto formato, conocido como *ARFF,* acrónimo de *Attribute-Relation File Format (Formato Fichero Atributo Relacional)*.

WEKA es un software de libre distribución desarrollado en Java. Está constituido por una serie de paquetes de código abierto con diferentes técnicas de preprocesado, clasificación, agrupamiento, asociación, y visualización, así como facilidades para su aplicación y análisis de prestaciones cuando son aplicadas a los datos de entrada

seleccionados. Estos paquetes pueden ser integrados en cualquier proyecto de análisis de datos, e incluso pueden extenderse con contribuciones de los usuarios que desarrollen nuevos algoritmos. Con objeto de facilitar su uso por un mayor número de usuarios, WEKA además incluye una interfaz gráfica de usuario para acceder y configurar las diferentes herramientas integradas.

Las características que hacen de WEKA una aplicación tan interesante son:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable. Está completamente implementado en Java y puede ejecutarse en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para preprocesamiento de datos, clustering, clasificación, regresión, etc.
- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.
- Proporciona acceso a bases de datos vía SQL gracias a la conexión JDBC (*Java Database Connectivity*) y puede procesar el resultado devuelto por una consulta hecha a la base de datos.

Un área importante que actualmente no cubren los algoritmos incluidos en Weka, es el modelado de secuencias. Por otro lado, otra carencia que presenta es que no permite realizar minería de datos multi-relacional, pero existen aplicaciones que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla que ya puede ser procesada con Weka.

2.4. Algoritmo de Clasificación: C4.5

El algoritmo C4.5 fue desarrollado por JR Quinlan en 1993, como una mejora del algoritmo ID3 desarrollado en 1986.

El algoritmo C4.5 genera un árbol de decisión a partir de los datos mediante particiones realizadas recursivamente. El árbol se construye mediante la estrategia de "primero en profundidad". El algoritmo considera todas las pruebas posibles que pueden dividir el conjunto de datos y selecciona la prueba que aporta la mayor ganancia de información. Para cada atributo discreto, se considera una prueba con *n* resultados, siendo *n* el número de valores posibles que puede tomar el atributo. Para cada atributo continuo, se realiza una prueba binaria sobre cada uno de los valores que toma el atributo en los datos. En cada nodo, el sistema debe decidir cuál de las pruebas escoge para dividir los datos.

Los tres tipos de pruebas posibles propuestas por el C4.5 son:

- La prueba "estándar" para las variables discretas, con un resultado y una rama para cada valor posible de la variable.

- Una prueba más compleja, basada en una variable discreta, donde los valores posibles son asignados a un número variable de grupos con un resultado posible para cada grupo, en lugar de para cada valor.

- Si una variable A tiene valores numéricos continuos, se realiza una prueba binaria con resultados A <= Z y A > Z, para lo cual debe determinarse el valor límite Z.

Todas estas pruebas se evalúan de la misma manera, mirando el resultado de la proporción de ganancia, o alternativamente, el de la ganancia resultante de la división que producen. Ha sido útil agregar una restricción adicional: para cualquier división, al menos dos de los subconjuntos C_i deben contener un número razonable de casos. Esta restricción, que evita las subdivisiones casi triviales, es tenida en cuenta solamente cuando el conjunto C es pequeño.

Las características del algoritmo C4.5 son:

- Permite trabajar con valores continuos para los atributos, separando los posibles resultados en 2 ramas A_i<=N y A_i>N.

- Los árboles son menos frondosos, ya que cada hoja cubre una distribución de clases, no una clase en particular.

- Utiliza el método "divide y vencerás" para generar el árbol de decisión final a partir de un conjunto de datos de entrenamiento.

- Se basa en la utilización del criterio de proporción de ganancia. De esta manera se consigue evitar que las variables con mayor número de posibles valores salgan beneficiadas en la selección.

- Es Recursivo.

2.5. Aplicación de aprendizaje automático en robótica

En esta sección se describen algunos trabajos realizados en robótica en los cuales se han aplicado técnicas de aprendizaje automático para mejorar el rendimiento de éstos.

a) Marvin

El robot usado fue un vehículo de radio control modificado [24]. Medía aproximadamente 45cm de largo, 25cm de ancho y 25cm de alto. Poseía cuatro ruedas

de 9cm de diámetro y un motor eléctrico que le dotaba de la energía suficiente para moverse.

Estaba dotado de parachoques delanteros y traseros para detectar los choques y cinco sónares con orientación: frente, atrás, izquierda, derecha y arriba.

Para el mapeo de los sensores de entrada apropiados para la salidas, se creó una red neuronal tal y como se muestra en la figura 2.7.

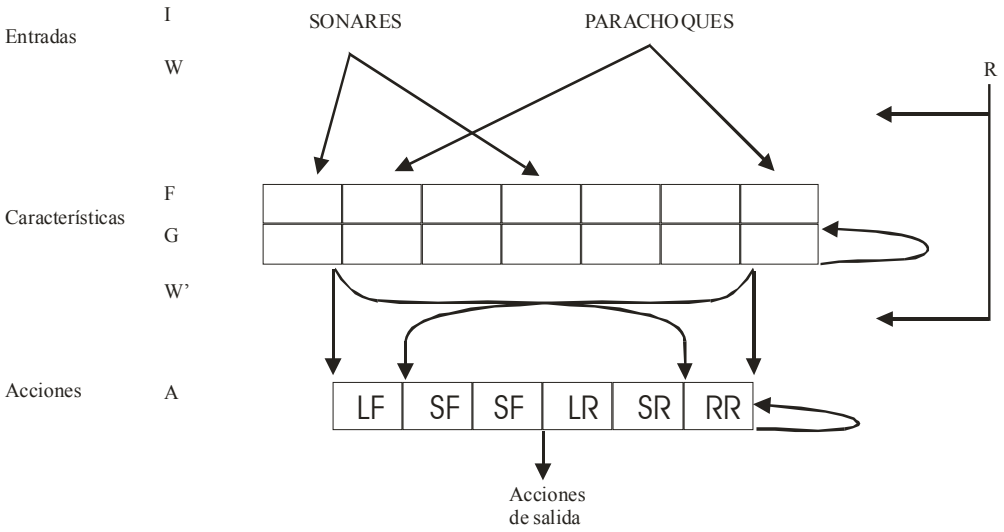

Figura 2.7: Marvin. Red para la selección de las acciones

Para el aprendizaje se realiza aprendizaje por refuerzo. En este proceso de aprendizaje se ajustan las matrices **W** y **W'** basado en el refuerzo de la información recibida.

El proceso se centra en asignar pesos a W y W' dado un estado particular de la red y una señal de refuerzo. La señal de refuerzo simplemente es decirle al robot si actúa bien o no, es el sistema de aprendizaje el que debe averiguar cómo incide en la toma de decisiones.

Si se considera la posibilidad de una primera señal de refuerzo positivo. Se debe aumentar la probabilidad de que Marvin ejecute las mismas acciones cuando la situación que se plantea sea idéntica. Para lograr esto, se fortalecen los pesos de la entrada de las características del nivel F de la arquitectura de la figura 2.7, con el fin de reconocer las mismas características. Además, para asegurarse de que se adopten las mismas medidas, se fortalecen los pesos de las características de la acción seleccionada.

Un comportamiento similar se tendrá si la señal de refuerzo es negativa. Con la salvedad de que en lugar de aumentar los pesos, estos serán disminuidos para intentar no seleccionar el mismo conjunto de acciones para situaciones similares.

Los resultados del aprendizaje se muestran en la tabla 1, en la cual se indican las acciones que se determinan realizar ante diferentes situaciones.

Entrada Sensores	Acción a realizar
Sónar izquierdo, distancia media	Avanzar recto
Sónar izquierdo, distancia lejana	
Sónar superior, distancia lejana	
Contacto de parachoques frontal	Retroceder recto
Sónar delantero, distancia cercana	
sónar superior, media distancia	
sónar superior, larga distancia	
sónar superior, larga distancia	Avanzar recto
Contacto de parachoques trasero	Avanzar a la derecha
Sónar delantero, larga distancia	Avanzar a la izquierda
Sónar delantero, distancia cercana	Retroceder a la izquierda
Sónar delantero, media distancia	
Sónar derecho media distancia	

Tabla 1: Marvin. Resultado del aprendizaje

b) Rodney

Este caso consistía en la generación de un control de los movimientos de las piernas de un robot con seis patas (figura 2.8). El control del movimiento de las piernas fue representado por una red neuronal (figura 2.9), mientras que la parametrización de dicha red se realizó aplicando algoritmos genéticos [24].

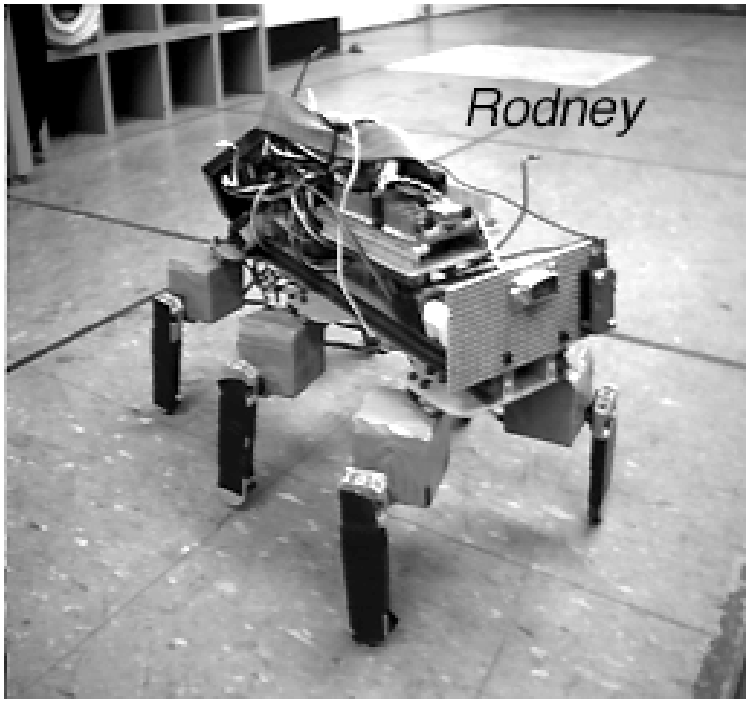

figura 2.8: Rodney, robot con seis patas.

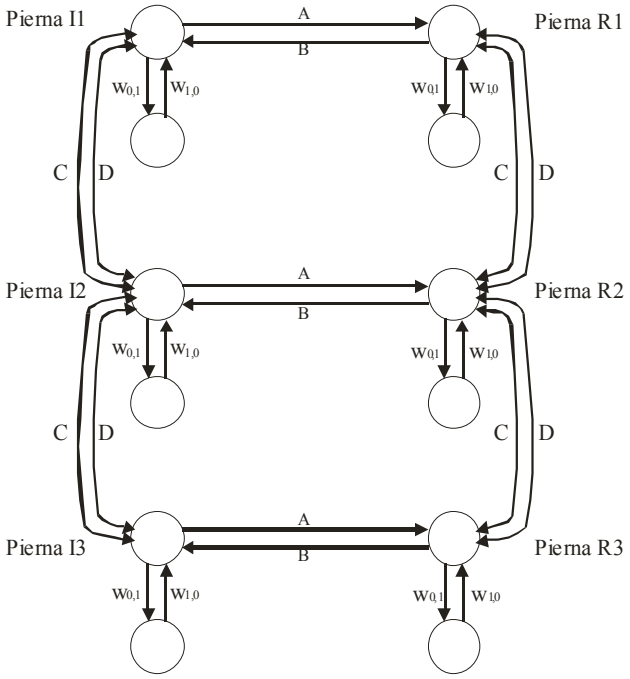

Figura 2.9: Rodney. Red neuronal para control de caminar

Para realizar el aprendizaje automático del robot, se aplicaron algoritmos genéticos para la calibración de los parámetros $w_{0,1}$, $w_{1,0}$, A, B, C y D. En la primera fase del algoritmo genético se calibran los parámetros $w_{0,1}$ y $w_{1,0}$ con una función de evaluación cuyos valores se muestran en la tabla 2.

Valor	Descripción
0	La salida de las dos neuronas son o <0 o >0
5	El estado de una neurona es >0 y el de la otra <0
10	El menor valor de las neuronas es >0 y cambia a <0
25	Valor de la nuerona de salida reducido a 0
40	El oscilador no converge a cero, pero la magnitud de la oscilación es < 1
41-59	La oscilación es cada vez más y más lenta
60	Salida de la neurora entre el rango de valores [0,1]

Tabla 2: Rodney. Función de fitness de la fase 1

Una vez terminada esta primera calibración, la cual abarcaba a cada pierna de manera individual, se aplicó otro algoritmo genético para calibrar los parámetros A, B, C y D, en este caso, necesarios para el comportamiento global del robot. La función de evaluación utilizada sigue la valoración que se muestra en la tabla 3, y dónde L es el número de centímetros que camina, T es el número de grados que gira mientras camina, y α es un número designado para distinguir el avance del robot.

Valor	Comportamiento
0	No hay movimiento ni oscilación en las piernas
10+L-(T/10)	Movimiento hacia delante
10+L- α (T/10)	Movimiento hacia atrás

Tabla 3: Rodney. Función de fitness de la fase 2

2.6. Robot Pioneer P3-DX

Los robots P3-DX [27] pertenecen a la familia de robots de movimientos desarrollados por Pioneer. Estos robots se caracterizan por poseer tracción diferencial, lo cual le dota de una alta movilidad y un manejo sencillo de los motores.

Las dimensiones del mismo se pueden ver en la figura 2.10.

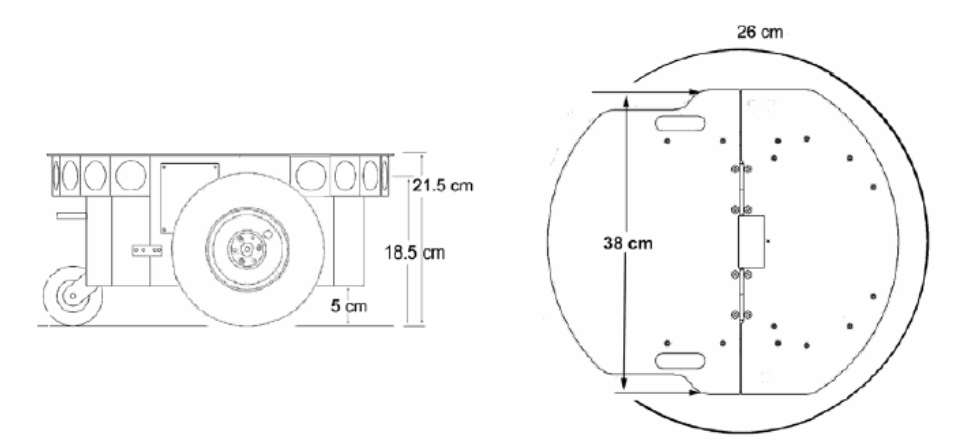

Figura 2.10: Dimensiones de Pioneer P3-DX

Este modelo de robot está dotado de parachoques y sónares. Estos elementos se explican a continuación:

- ***Parachoques*:** Son sensores de contacto todo o nada; es decir lanzan una señal cuando son presionados (por ejemplo al chocar con un obstáculo). Dispone de diez parachoques distribuidos cinco en la parte delantera y otros cinco en la parte trasera del robot, distribución que se puede ver representada en la figura 2.11.

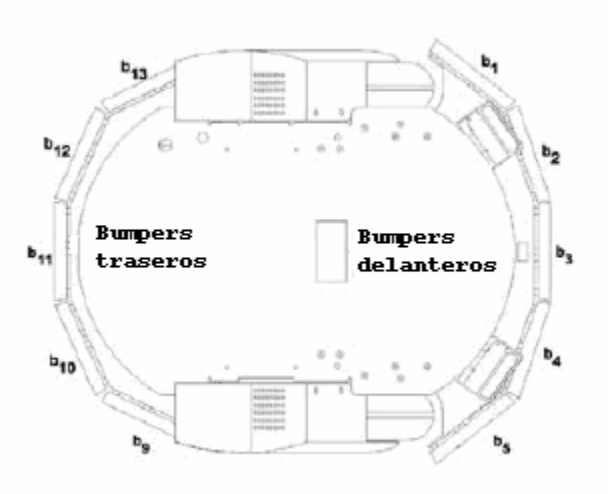

Figura 2.11: Parachoques

- ***Sónares*:** Estos sensores permiten detectar obstáculos antes de que el robot llegue a chocar con ellos. Los sónares emiten señales de sonido para determinar si hay obstáculo o no dependiendo de si recibe eco o no de la señal emitida. Los Pioneer P3-DX poseen seis sónares delanteros (1-6), así como otros dos que controlan los laterales del robot (0 y 7). La distribución de los sónares se puede observar en la figura 2.12.

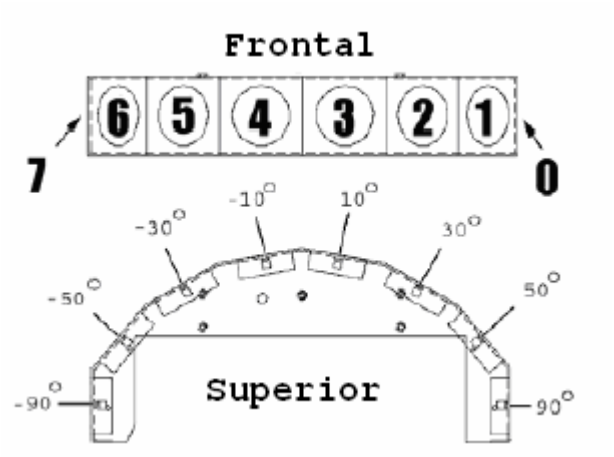

Figura 2.12: Sónares

Como todos los elementos electrónicos, estos robots también disponen de baterías para poseer una cierta autonomía energética. Estas baterías se colocan en la cavidad habilitada para ello en la parte trasera de robot, tal y como se puede observar en la figura 2.13.

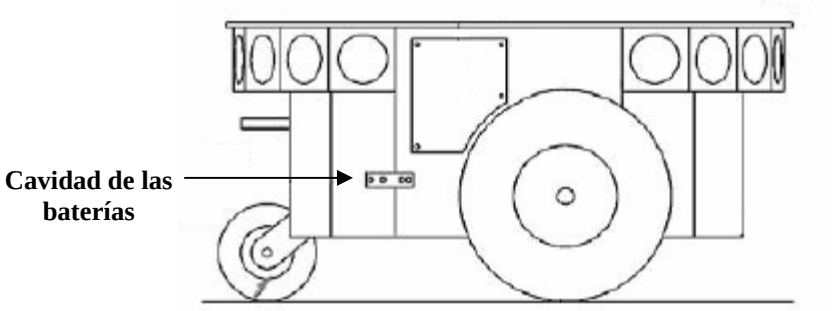

Figura 2.13: Cavidad de baterías

Para conocer el nivel de carga de batería, se dispone de un led, el cual posee un color verde cuando la carga es completa. Este color irá variando de naranja hasta rojo cuando el nivel de batería es demasiado bajo. El indicador mencionado estará contenido dentro del panel de control; panel de control que tal y como se muestra en la figura 2.14 posee información y botones básicos del robot.

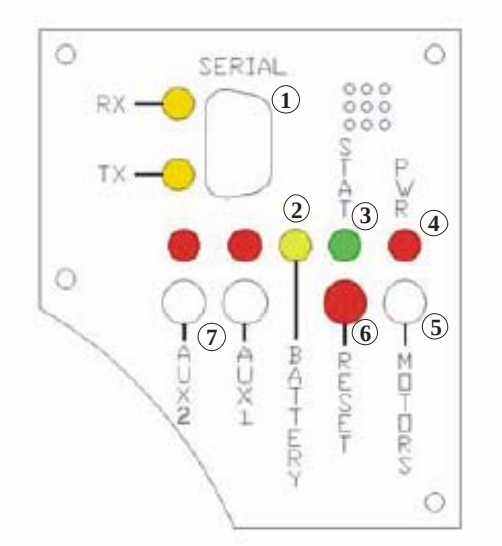

Figura 2.14: Panel de control

Estos botones indican:

1. **Serial:** Puerto de conexión serie, a través del cual se realiza la conexión con el robot. Posee dos indicadores: RX (indicador de entrada de datos) y TX (indicador de salida de datos).
2. **Batery:** Indica el nivel de carga de la batería del robot. Verde carga completa, y degradado de naranja a rojo a medida que la carga disminuye.
3. **Stat:** Estado del robot. Parpadea lentamente cuando está a la espera de una conexión con el cliente. Si conecta con el cliente, ya sea un controlador o el joystick parpadeará rápidamente.
4. **Pwr:** Indicador de encendido.
5. **Motors:** Botón que cambia el estado de los motores, es decir si están habilitados los deshabilita y viceversa.
6. **Reset:** Resetea el robot.
7. **Aux1/Aux2:** Indicadores de conexiones auxiliares.

Estos robots están dotados de un sistema de tracción diferencial, por el cual el robot posee dos ruedas motrices (ruedas conectadas a motores para conseguir movimiento) y una rueda libre para conseguir mantener la estabilidad. Con este sistema, se consigue que las ruedas roten con velocidades distintas, lo que causa que el robot pueda girar, por ejemplo para girar a la derecha la rueda izquierda debe poseer más velocidad, y viceversa para girar hacia el lado opuesto.

Las ventajas de poseer este sistema son:

- Posee un diseño sencillo.
- Es fácil de implementar.

- Permite una gran maniobrabilidad.

La desventaja reseñable que presenta es la dificultad de hacer ir al robot en línea recta, y que esta situación conllevaría la misma velocidad en las ruedas y que dichas velocidades se iniciaran al mismo tiempo.

2.7. Simulador MobileSim

MobileSim es una aplicación para la simulación de las plataformas "MobileRobots/ActivMedia" y sus entornos, para la depuración y experimentación con ARIA (Librería genérica para el control de la familia de robots de movimientos desarrollado por Pioneer.).

MobileSim está construido sobre el simulador "Stage", creado por Richard Vaughan, Andrew Howard y otros participantes en el proyecto, añadiendo algunas modificaciones para "MobileRobots" (familia de robots de movimientos de Pioneer). MobileSim aun está en desarrollo, por lo que hay características que aun no están desarrolladas; las limitaciones que posee actualmente son:

- No tiene en cuenta la reflexión de los sónares.
- No tiene disponible pinzas de agarre.
- No dispone de parachoques.
- No contempla los sensores infrarrojos.
- No implementa puertos de entrada/salida analógica o digital.
- No contempla la inclusión de radio balizas.
- No se han desarrollados sensores de detección de sustancias.
- No permite la inclusión de cámara de video.
- Si bien es posible hacer correr la simulación más rápido (o mucho más lento) que el "tiempo real", no está bien probado cómo afecta esto a las aplicaciones clientes. Muchos clientes poseen límites de tiempo basado en el sistema de tiempo real, o bien, puede que no sean capaces de procesar los paquetes con la suficiente rapidez, lo que conlleva perdida de datos.

MobileSim, para simular las paredes y demás obstáculos del entorno, utiliza la información proporcionada por mapas para "MobileRobots" (ficheros '.map'). Para la generación de estos ficheros se pueden utilizar las aplicaciones:

- **Mapper3:** Aplicación que permite crear y editar ficheros '.map' que contienen mapas útiles para el simulador MobileSim. Permite crear el espacio de simulación, así como guardar mapas como imágenes de mapa de bits, y con un servidor adecuado, cargar y descargar mapas automáticamente. Una característica reseñable de esta aplicación es permitir utilización de láser para convertir la lectura de estos sensores en un mapa.

- **Mapper3Basic:** Simplificación de Mapper3, en cuya simplificación no se incluye la utilización de sensores láser. Se mantiene el resto de utilidades como las de agregar líneas, establecer puntos concretos, inclusión de otros objetos, etc., con el fin de definir el espacio de simulación.

Las funcionalidades que ofrece este simulador son:

- Simulación de sensores sónares.
- Estimación de odometría (con acumulación de errores).
- Múltiples robots (desempeño adecuado es posible en un ordenador rápido con más de 50 robots).
- El entorno y los obstáculos son cargados desde un archivo (mapa).
- Los objetos (robots u obstáculos) pueden ser configurados para ser detectables sólo por algunos sensores.
- Guardar imágenes "instantáneas" de la simulación o guardar una secuencia de imágenes.
- Los ficheros de mapas pueden ser recargados en tiempo de ejecución.
- Los robots pueden reposicionarse interactivamente en el entorno.
- Opciones de visualización para los sensores, robots y los obstáculos.

Los modelos de robots de movimientos que tiene preconfigurado el simulador son: Pioneer P3 DX, Pioneer AT, PowerBot, AmigoBot, PeopleBot, Patrolbot, Seekur, Pioneer 2 y Pioneer 1. Los parámetros tanto de estos modelos de robots, como de los dispositivos que poseen, pueden ser personalizados en un archivo de configuración. La inclusión de nuevos modelos puede ser definida sobre la base de modelos existentes.

Los planes futuros incluyen para los sónares mejores modelos, la simulación de dispositivos adicionales, y diversas características de interfaz de usuario para la creación de nuevos robots simulados, descarga de mapas, y ajustar sus propiedades.

MobileSim ha sido probado en Windows 2000, Windows XP, RedHat GNU / Linux 7.3 y Debian GNU / Linux 3.1.

Este simulador es de libre distribución, se puede encontrar en http://robots.mobilerobots.com/MobileSim, y para instalarlo tan solo es necesario seguir las instrucciones que se proporcionan.

CAPÍTULO 3: **OBJETIVOS**

Un robot de movimiento autónomo eficiente es aquel capaz de moverse de manera genérica por cualquier entorno y no solo por aquellos mundos controlados. En el caso de limitar el movimiento a espacios controlados el problema se reduce en gran medida, realizándose habitualmente en estas situaciones programaciones de movimientos preestablecidas para conseguir la navegación por dicho entorno.

Sin embargo para conseguir que estos robots de movimiento sean capaces de desplazarse por entornos desconocidos es necesario que superen situaciones inesperadas para poder ser eficaces.

Para conseguir esta autonomía en los movimientos del robot, se debería comenzar por desarrollar los niveles de control más bajos y básicos. En el caso de los robots de movimiento, estos niveles más bajos deberían evitar chocar ante los obstáculos que encuentre en su camino.

Pero, para que el robot sea útil, no solo debería moverse sin chocar, sino que debe realizar alguna tarea más compleja o específica. Al realizar esta tarea el robot no debería olvidar estos “instintos primarios” para evitar el obstáculo.

- **Objetivo General:**

Desarrollar una arquitectura de control, siguiendo el paradigma reactivo, para los robots P3-DX de la casa Pioneer. En concreto se pretende realizar una arquitectura de subsunción y cuyos comportamientos reactivos que la componen, controlen y doten de funcionalidad a dichos robots.

Las capacidades de las que se dotará a estos robots serán orientadas a la navegación de manera autónoma de los mismos.

- **Objetivos Específicos:**

    1. Crear pruebas independientes para la validación de las funcionalidades que aportan los robots Pioneer P3-DX.

    2. Establecer como nivel básico en la arquitectura la capacidad de recuperación del robot ante un caso extremo como puede ser un choque.

    3. Dotar a los robots de la funcionalidad para evitar los obstáculos que encuentre a su paso.

    4. Dar funcionalidades de navegación tales como recorrer un pasillo o una planta de un edificio.

    5. Aplicar técnicas de aprendizaje automático para “refinar” comportamientos.

    6. Desarrollar una página Web que recopile toda la información del trabajo realizado en el presente proyecto fin de carrera, para facilitar el uso del mismo en futuros trabajos.

CAPÍTULO 4: **MEMORIA TRABAJO REALIZADO**

En el presente capítulo se detalla el trabajo que se ha realizado durante el desarrollo del proyecto fin de carrera *"Una Arquitectura de Subsunción para robots Pioneer P3-DX".*

Los puntos que se tratan durante el presente capítulo son: explicación sobre las pruebas realizadas para la validación de las funcionalidades del robot Pioneer P3-DX, estudio de la arquitectura de subsunción desarrollada, se explica el trabajo realizado mediante el uso de aprendizaje automático, y finalmente se comenta la Web realizada como fuente de recopilación de información.

4.1. Pruebas del robot Pioneer P3-DX

Para comenzar a conocer las funcionalidades que aportan los robots Pioneer P3-DX se inició el proyecto realizando unas pruebas básicas sobre los diferentes dispositivos que poseen estos robots, y conseguir, adicionalmente, una familiarización con el API específica para este modelo de robot, desarrollada con anterioridad mediante un trabajo dirigido.

Las pruebas desarrolladas con dicha finalidad han sido las siguientes:

- ***Prueba de motores***

Esta primera prueba tiene como cometido interactuar con los motores del robot. Para ello se han establecido una serie de movimientos prefijados a ejecutar por el robot. Los movimientos que se han preprogramado han sido:

1. Establecer una velocidad de rotación de 100 grados/segundos.

2. Eliminar la velocidad de rotación anterior estableciéndola a 0.

3. Establecer una velocidad de 300 mm/seg a la rueda izquierda y de 100 mm/seg a la derecha. Con esto se consigue que el robot rote hacia la derecha. Esta situación se mantiene durante cinco segundos.

4. Indicar que se mueva un metro hacia delante.

5. Indicar que se mueva un metro hacia atrás.

6. Establecer una velocidad lineal de 200 mm/seg.

7. Parar el robot.

8. Establecer una velocidad de 0 mm/seg a la rueda izquierda y de 200 mm/seg a la derecha. Con esto se consigue que el robot rote hacia la izquierda.

9. Indicar al robot que rote a 50 grados/segundo. Girará hacia la izquierda.

10. Indicar al robot que rote a -50 grados/segundo. Girará hacia la derecha.

Para conseguir que todos estos movimientos se realicen y no se solapen unos con otros, se han hechos excluyentes mediante el bloqueo del robot para que no reciba ninguna otra orden hasta que la tarea finalice.

Adicionalmente se estableció un tiempo de 3 segundos de espera entre la emisión de una acción a realizar y la siguiente. El motivo de la inclusión de este tiempo de espera para la emisión es conseguir hacer visible con mayor facilidad el cambio de una tarea y otra.

- ***Validación de* parachoques**

El objetivo de esta prueba es el de validar los parachoques del robot. El funcionamiento consiste en indicar al robot que se mueva hacia delante en línea recta y en el caso de encontrar algún obstáculo chocará con él. Una vez que el robot choca se activará uno de los parachoques delanteros, y el robot se moverá hacia atrás girando en sentido opuesto al obstáculo para después, avanzar de nuevo intentando de esta manera evitar el obstáculo.

En esta prueba se recuperan las lecturas de los parachoques, tanto delanteros como traseros. Para conseguirlo es necesario obtener el estado del robot, y aplicar máscaras para conseguir tan solo la información de los parachoques que se desea obtener, ya sean delanteros o traseros.

En la prueba no es suficiente con conocer si se produce un choque delantero o trasero, sino que también es necesario saber qué parachoques es el que choca. Esto es necesario para poder conocer la posición del obstáculo con respecto al robot.

- ***Control teleoperado***

Esta validación tiene como finalidad manejar el robot de manera teleoperada, es decir, dirigido por el teclado del ordenador conectado a él.

Ha sido necesario crear un controlador para recibir las señales del teclado, de las teclas relevantes al control. Cuando una de estas teclas es pulsada, se realizará la modificación de la velocidad del robot que corresponda en cada instante: aumento o disminución de la velocidad, ya sea velocidad lineal si son las teclas de avance o retroceso o velocidad rotacional si se pulsa las teclas de giro.

Los controles de teclado para el manejo teleoperado se muestran en la figura 4.1.

Figura 4.1: Controles para control teleoperado

Cabe destacar que las velocidades son incrementales al pulsar las teclas, por ejemplo si se pulsa la tecla de avanzar hacia delante, se produce un incremento de la velocidad (si estaba parado, ahora tiene una velocidad de $\triangle x$). Esta velocidad permanecerá constante hasta que se pulse de nuevo la tecla de avance, en cuyo caso volverá a aumentar la velocidad, o se pulsa la tecla de retroceso con lo que se produciría un decremento velocidad, o como ultima opción pulsando la barra espaciadora que pondrá la velocidad del robot a cero. El incremento o decremento de velocidad será un valor por defecto establecido en la prueba.

En un principio, este modo de ejecución puede ser más útil para el simulador, ya que para la ejecución en el robot físico sería engorroso, debido a que debe existir un cable que conecte el PC con el robot, con lo cual la movilidad de este último será reducida.

- ***Seguimiento de contorno***

En esta prueba se pretende hacer que el robot siga el contorno de una habitación en sentido antihorario. Este seguimiento de contorno se realiza con los sensores sónares derechos del robot.

Para ello se tienen dos componentes, el primero de ellos se encarga de hacer que el robot se mueva y detectar si encuentra de frente algún obstáculo, y está lo suficientemente cerca (esta distancia se puede establecer). En el caso de detectar el obstáculo el robot girará para evitarlo.

El segundo componente se encarga de mantener el robot a una cierta distancia de la pared. Para ello, hace uso de los sónares situados a la derecha del robot (por ello se ha dicho anteriormente que se movería en sentido antihorario). Si la distancia a la pared es distinta a la establecida (esta distancia puede ser modificada en los argumentos del comportamiento) se rectificará la orientación del robot. Para conseguir que la rectificación del robot se produzca de manera suave se emplea la siguiente función:

```
X = distancia_pared - lectura_sonares_derecha

Y = X/10
```

Consiguiéndose una distribución lineal tal y como se muestra en la figura 4.2.

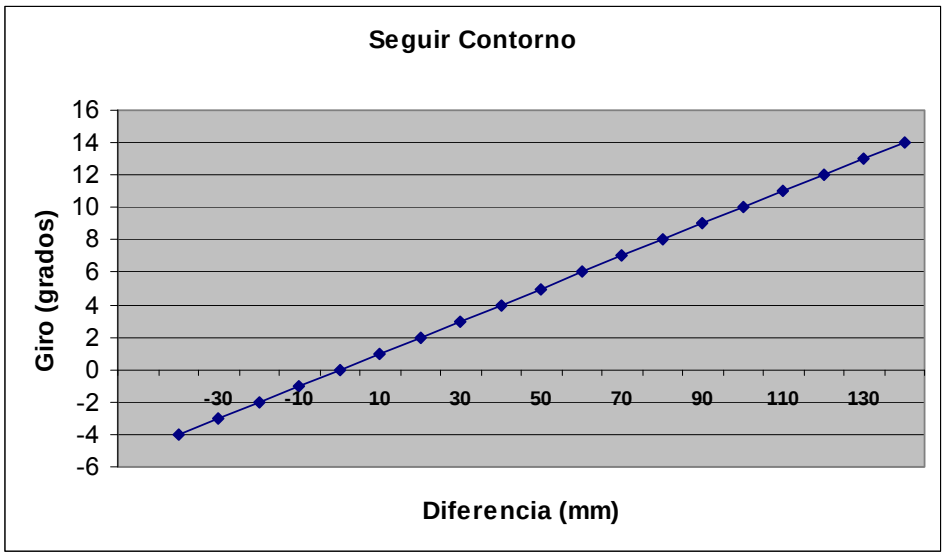

Figura 4.2: Gráfica de distribución para seguimiento de contorno

Como se aprecia en la figura 4.2, a distancias mayores se consiguen cambios mayores de orientación y a distancias menores cambios más suaves. Por el mero hecho de hacer la diferencia de la distancia establecida a la pared menos la lectura de los sónares de la derecha, se consigue que el sentido de la rectificación sea el correcto, es decir, si la lectura de los sónares indica que está más separado de la pared de lo

indicado, el sentido a rectificar será negativo con lo cual se girará hacia la derecha acercándose a la pared tal y como se deseaba, y del mismo modo se obtendrá el sentido deseado si la distancia establecida a la pared es mayor que la leída por los sónares.

Este segundo componente, también se encarga de detectar que en el caso de que los sónares de la derecha no detecten nada, se está ante una esquina, con lo cual deberá girar el robot para establecer de nuevo la referencia a la pared. Para ello se va girando poco a poco a la derecha el robot hasta que encuentre una lectura del sónar.

4.2. La arquitectura de subsunción para robot Pioneer P3-DX

En el presente apartado se expone de manera global la arquitectura desarrollada para los robots Pioneer P3-DX. Una vez vista de manera global dicha arquitectura, se procede a explicar en mayor profundidad cada uno de los niveles que la componen, así como los comportamientos que integran cada uno de ellos.

4.2.1. Descripción de la arquitectura

La arquitectura de subsunción final que se ha desarrollado para el robot de movilidad Pioneer P3-DX se corresponde con la figura 4.3.

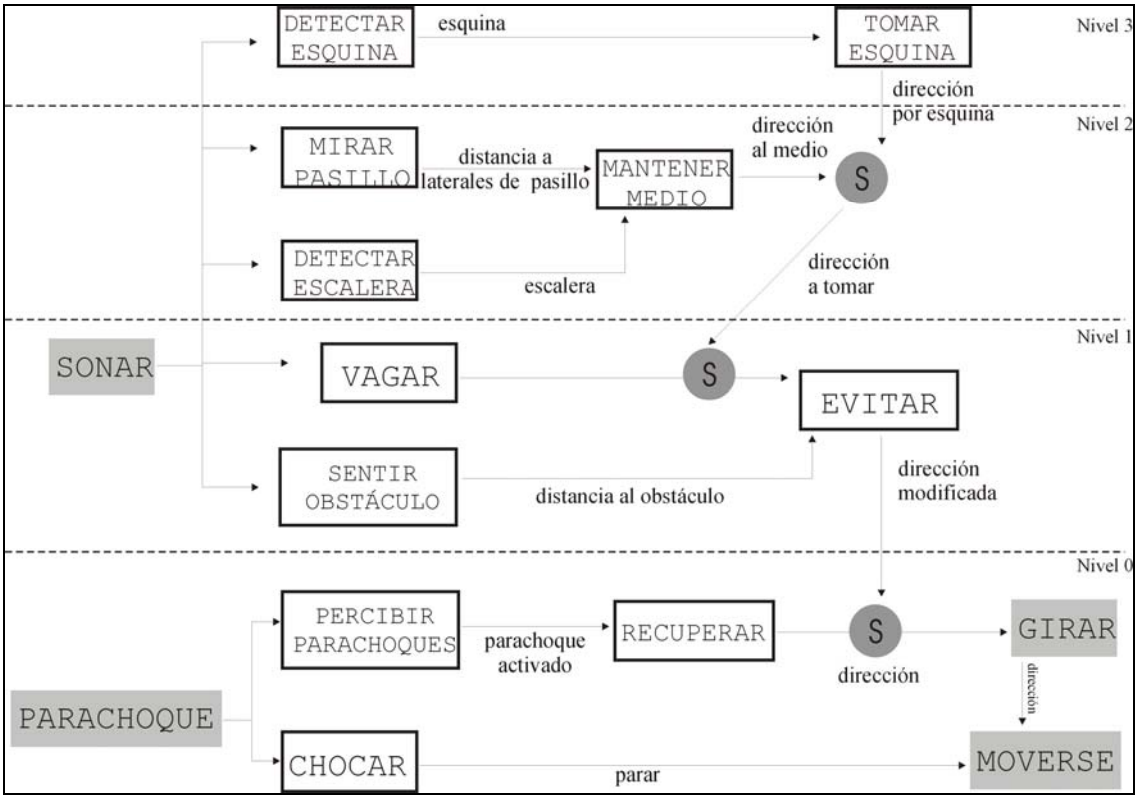

Figura 4.3: Arquitectura de Subsunción desarrollada

Como se puede observar en la figura 4.3, la arquitectura está compuesta por cuatro niveles. Estos niveles se pueden clasificar en dos grandes grupos según la naturaleza del comportamiento que llevan a cabo:

- **Básicos**: abarca los dos niveles inferiores. Estos niveles son los que comprenden lo que se puede denominar "instintos del robot", y que buscan mantener la supervivencia del robot, supervivencia que en el caso actual se corresponde con evitar chocar con obstáculos y en caso de chocar con ellos conseguir recuperarse del choque.

- **De alto nivel**: supondrán los dos niveles superiores. En este caso se encargan de darle una funcionalidad concreta al robot.

Como se ha comentado en el apartado 2.1.2 del presente documento, los niveles se ejecutan de manera concurrente, y en el caso de que varios produzcan salida para la misma situación del entorno, los comportamientos de las capas superiores subsumirán a los correspondientes de la capa que se encuentra por debajo de ellas.

Por otro lado, la arquitectura refleja como la información que reciben los diferentes niveles de comportamientos es la proporcionada por los sensores. En el caso del robot P3-DX de pioneer con el que se ha trabajado, los únicos sensores de los que se disponen son sónares y parachoques (delanteros y traseros).

Las acciones que permite realizar el robot son todas de movimientos, por lo que los actuadores sobre los que los comportamientos pueden actuar son los motores, para de esta manera modificar su posición y orientación respecto el entorno que le rodea. Para cambiar su posición y orientación se "jugará" con las velocidades impuestas a cada uno de los motores conectados a las ruedas, por ejemplo si la rueda izquierda tiene mayor velocidad que la rueda derecha, el robot girará a la derecha.

Aunque los diferentes niveles de la arquitectura son prácticamente independientes, hay situaciones en las que sí es necesario comunicarse.

A continuación se describe cada uno de los que conforman la arquitectura de subsunción presentada en la figura 4.3.

4.2.2. Nivel 0

En este nivel, se determina cuándo un robot choca y cuándo no, haciendo uso de los sensores de contacto todo o nada (parachoques) que dispone el robot. En el caso de que se produzca un choque, el robot se recuperará de esta situación para poder seguir moviéndose.

Se ha incluido este nivel como el más bajo debido a que es la situación más extrema que se puede encontrar, ya que lo ideal es que el robot no chocara con ningún obstáculo que encontrara en su camino, pero en el caso de que chocara deberá recuperarse ante estas situaciones indeseadas.

4.2.2.1.Comportamientos

En este nivel se pueden encontrar dos comportamientos: chocar y recuperar.

- **Chocar:**

Cuando se dé el caso de que se produzca un choque, uno de los parachoques lanza una señal de activación. Al producirse ese choque, los motores inmediatamente se pararán para evitar forzar el movimiento en un sentido en el que no es posible debido a la existencia de un obstáculo.

- **Recuperar:**

La labor de este comportamiento es la de recuperar el movimiento del robot ante una situación extrema como es un choque. Para ello es necesario realizar un procesado de las señales de los parachoques para determinar cuál de ellos es el que ha chocado y por tanto determinar la posición relativa del obstáculo respecto del robot. La salida de este procesamiento será utilizada por el comportamiento para evitar el obstáculo moviéndose hacia delante o hacia atrás dependiendo de si el parachoques era trasero o delantero, y girando para de este modo conseguir que el robot pueda seguir con su movimiento pero con una orientación distinta evitándose un nuevo choque con el obstáculo.

4.2.2.2.Pruebas en el simulador

Las únicas pruebas que se han realizado han sido en el robot real, ya que el simulador utilizado, *MobileSim*, no contempla la funcionalidad para parachoques y no lanza dichas señales.

4.2.2.3.Pruebas en entorno real

En las pruebas realizadas en el robot real, el 100% de las ocasiones de choque eran detectadas por los parachoques.

Cada ocasión en la que se detectaba el choque, el robot invierte el sentido de movimiento durante un periodo de tiempo para solventar el obstáculo.

4.2.2.4.Pseudocódigo

```
Chocar{
      choque_delantero=ObtenerLecturaBumpersDelanteros();
      choque_trasero=ObtenerLecturaBumpersTraseros();
      Si (estaEjecutandoChoqueAnterior){
            ContinuarRectificacion();
      }
      Si no{
            Si (chocaBumperDelantero(choque_delantero)){
                  ActualizarVelocidad(0);
            }
            Si (chocaBumperTrasero(choque_trasero)){
                  ActualizarVelocidad(0);
            }
      }
}

Recuperar{
      parachoques_activo(percibirParachoques());
      si (estaEjecutandoChoqueAnterior){
            ContinuarRectificacion();
      }
      si no{

            si(esParachoqueDelantero(parachoqueActivo)==true){
                  anguloChoque=calcularAngulo(parachoques_activo);
                  modificarOrientacion(orientacionRobot, anguloChoque);
                  actualizarVelocidad(-velocidadRectificacion);
            }
            si no{
                  si (esParachoqueTrasero(parachoqueActivo)==true){
                        anguloChoque=calcularAngulo(parachoques_activo);
                        modificarOrientacion(orientacionRobot,
                        anguloChoque);
                        actualizarVelocidad(velocidadRectificacion);

                  }
            }
      }
}
```

Como se puede observar en el pseudocódigo, un choque viene determinado por la activación de algunos de los parachoques (delantero o trasero). Si uno de los parachoques se activa, la velocidad del robot se establece a 0 para una posterior recuperación.

A la hora de realizar la recuperación ante un choque en primer lugar debe detectarse el parachoques que se activó para calcular a partir de éste el ángulo del choque. Una vez calculado el ángulo de choque se obtiene la nueva orientación del robot así el sentido de la velocidad a establecer, negativo en caso de que el choque se produzca con un parachoques delantero y positivo en el caso contrario de que el choque se produzca con un parachoques trasero.

4.2.3. Nivel 1

Para este segundo nivel de la arquitectura se han desarrollado dos capacidades para el robot envueltas en la misma capa de la arquitectura, la primera consiste en detectar situaciones en las que el robot podría chocar, mientras que la segunda permite al robot moverse (vagar). Estas dos capacidades se han incluido en el mismo nivel ya que son capacidades que se complementan la una a la otra y no entran en conflicto entre sí, es decir, al vagar se dota al robot de una velocidad para que se mueva sin tener en cuenta la orientación, solo moverse; mientras que al detectar el obstáculo lo que se realiza es modificar esa orientación con la que se mueve.

4.2.3.1.Comportamientos

- **Evitar:**

Para la detección de obstáculos se utilizan los sónares que posee el robot. Con ellos se detectarán los obstáculos que se encuentran a determinadas distancias, pudiendo de este modo predecir cuándo se producirá un choque y cuándo no. Hay que tener en cuenta otros factores como la posición del obstáculo respecto al robot para determinar hacia dónde debería moverse y conseguir de este modo evitarlo.

Ya que la calibración de estos sensores para determinar cuándo se producirá un choque y cuándo no es muy complicada, se ha aplicado aprendizaje automático para discriminar, a partir de la propia experiencia del robot y del valor de las lecturas de los sónares, qué situaciones son susceptibles de choque y cuáles no. El trabajo realizado en este campo se describirá detalladamente en la sección 4.3. A modo de resumen, se puede indicar que se genera un árbol de decisión para determinar en qué circunstancias el robot chocará o en que situación no, aplicando el algoritmo C4.5 (J48 de Weka).

El árbol de clasificación definitivo se creó a partir de los datos obtenidos en el simulador *MobileSim* en cinco escenarios distintos y teniendo en cuenta la superficie de los parachoques que aumentan la planta del robot.

El preprocesado de datos se realizó para una predicción de choque de tres instantes de tiempo, es decir que un choque que se produce en el instante de tiempo 15 será predicho en el instante 12, consiguiéndose con ello darle tiempo suficiente al robot para girar satisfactoriamente y evitar el obstáculo.

Con estas características se obtiene un árbol de decisión cuyas especificaciones se muestran en la tabla 4.

```
Número de hojas del árbol: 138

Tamaño del árbol: 275 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 97.2583%

 === Matriz de confusión ===

     a     b   <-- clasificado como
  6730   509 |     a = SI CHOCA
   408 25800 |     b = NO CHOCA
```

Tabla 4: Características del árbol de decisión para la detección de obstáculos

Con esta técnica de Aprendizaje Automático se consigue detectar los obstáculos con los que chocaría el robot y evitarlo mediante un cambio de orientación del mismo.

- **Vagar:**

El comportamiento vagar, que comprende al nivel 1 de la arquitectura, debe permitir al robot pasear sin un rumbo específico. Para conseguirlo se debe actuar sobre los motores para controlar la velocidad y mantener en movimiento el robot. Este comportamiento se complementará con la detección de obstáculos comentada anteriormente, para conseguir mantener el movimiento del robot sin que se produzcan choques.

Pero, ¿cómo debería controlar la velocidad? En el presente proyecto, la decisión que se toma para responder a esta pregunta ha sido: si los sónares no detectan obstáculos cercanos, se aumenta la velocidad del robot; pero a medida que el robot se acerca hasta un obstáculo (una detección del sónar) la velocidad irá disminuyendo. Hay que destacar que el aumento de la velocidad no se realiza de manera continua, sino que para evitar que el robot alcance una velocidad demasiado elevada, habrá una cota máxima de velocidad la cual una vez alcanzada no se sobrepasará; esta cota máxima estará prefijada en el comportamiento.

A continuación se puede ver un ejemplo de esta modificación de la velocidad:

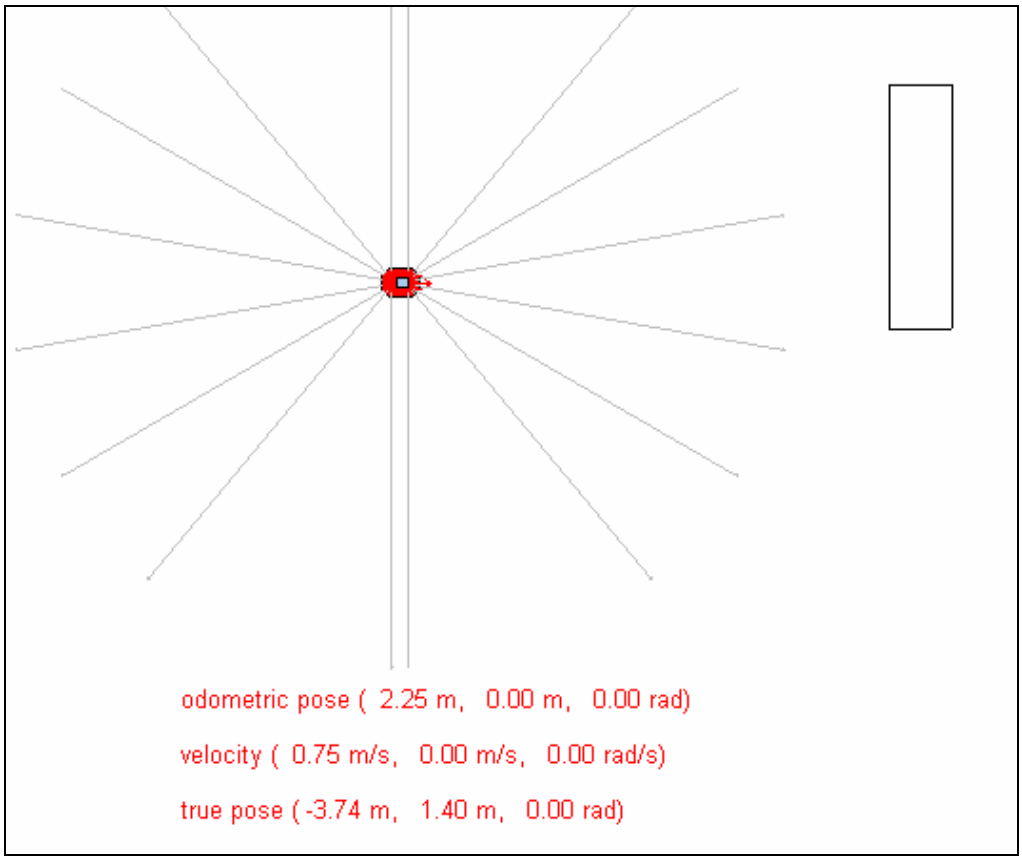

Figura 4.4: Control de velocidad con el robot alejado al obstáculo

En la figura 4.4 obtenida del simulador *MobileSim*, se puede apreciar como al no encontrar ningún obstáculo, la velocidad que alcanza el robot es de 0.75 m/s, siendo esta la máxima velocidad permitida en la ejecución.

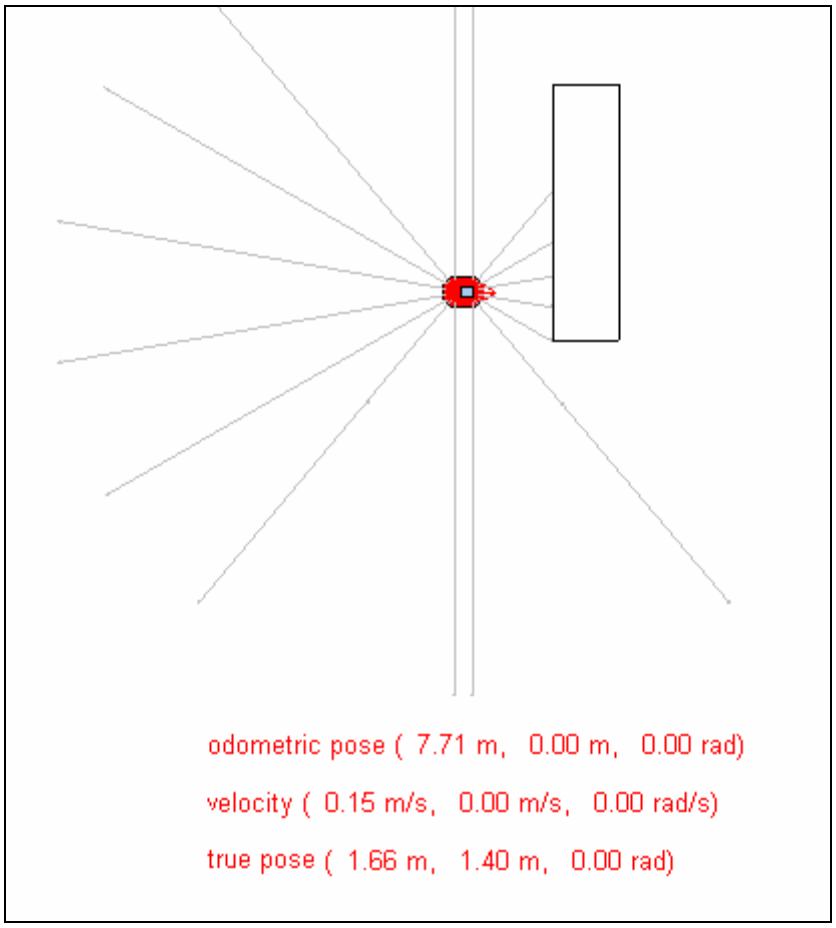

Figura 4.5: Control de velocidad con el robot cercano al obstáculo

En la figura 4.5 extraída de la misma ejecución en el simulador, se encuentra al robot muy cercano a un obstáculo, por lo que se puede apreciar como la velocidad es mucho menor, 0.15 m/s.

Si se mantiene la ejecución de la prueba simulada, se puede ver la actuación completa del nivel 1 de la arquitectura, y cómo actúa esquivando el obstáculo. Lo que sucede es que se detecta un obstáculo con el que chocaría según el árbol de decisión desarrollado, por ello se cambia la orientación del robot, y de este modo se consigue que el robot se mueva en otra dirección y así siga con su movimiento sin que choque con ningún obstáculo. El giro a realizar por parte del robot se incluyó por defecto en el comportamiento, estableciéndose a un valor de 80 grados. Sin embargo, el sentido del giro variará según el obstáculo lo detecte un sónar u otro, es decir si el objeto con el que chocaría el robot fue detectado por un sónar que se encuentra en el lado derecho del robot, el obstáculo se encontrará a este mismo lado con respecto al robot, con lo cual se girará hacia la izquierda, y girará a la derecha justo en el caso contrario. Este hecho se puede ver reconstruido a continuación:

En figura 4.6 se muestra cómo el robot va a chocar con un obstáculo. Se puede observar como son los sónar situados a la izquierda del robot los que detectan el obstáculo (se encuentra marcado en rojo la señal de los sónares que interceptan).

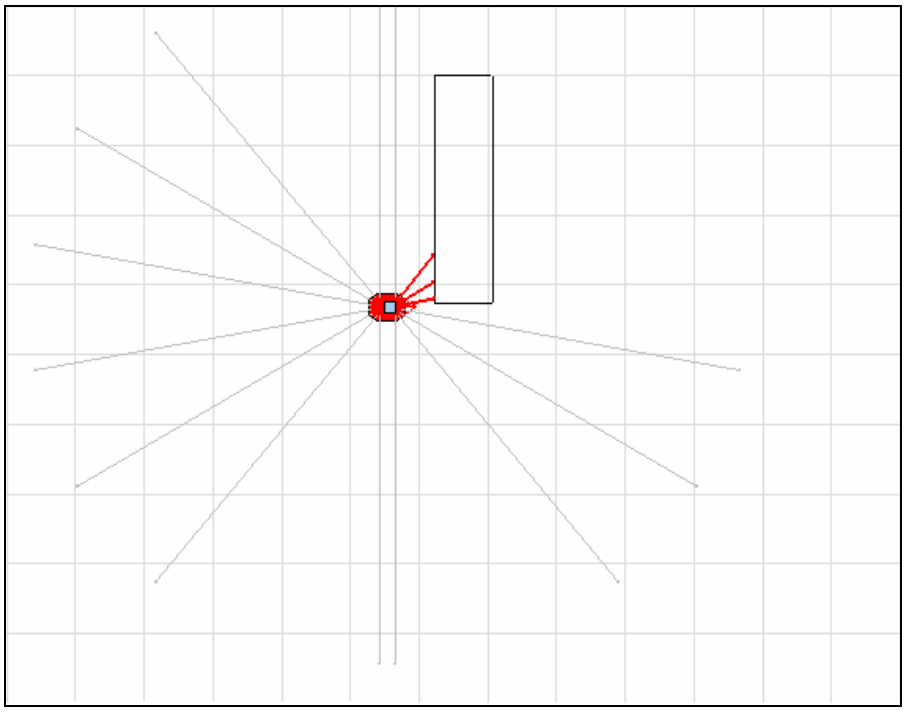

Figura 4.6: Detección de un obstáculo situado a la izquierda del robot

Para evitar el obstáculo el robot girará a su derecha tal y como se muestra en la figura 4.7.

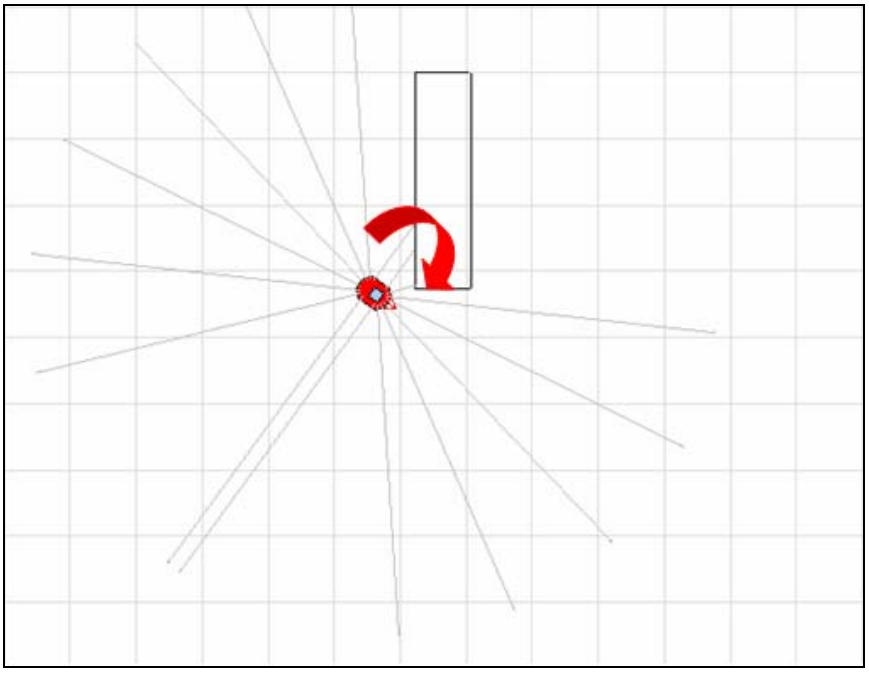

Figura 4.7: Giro del robot al encontrar un obstáculo

Por lo tanto se puede observar como el robot al modificar su orientación puede seguir su movimiento sin producirse un choque con el obstáculo.

4.2.3.2.Pruebas en el simulador

En primer lugar, es reseñable que en las pruebas realizadas sobre la detección de obstáculos, tan solo se comentan los resultados obtenidos con el árbol de decisión definitivo que se ha utilizado en el comportamiento. El resto de pruebas realizadas con otros árboles quedarán detalladas en la sección 4.3 del presente documento.

Probando este árbol en el simulador se obtuvieron buenos resultados. Se obtuvo un porcentaje del 2,57% de choques sobre el total probado. Esto se corresponde con 8 choques de 311 situaciones de posibles choques.

Con el control de los motores se consigue que la velocidad del robot sea menor cuanto más cercano está al obstáculo.

En la figura 4.8 se muestra gráficamente la relación entre la velocidad de movimiento del robot y la distancia a la que se encuentra del obstáculo durante la ejecución en el simulador.

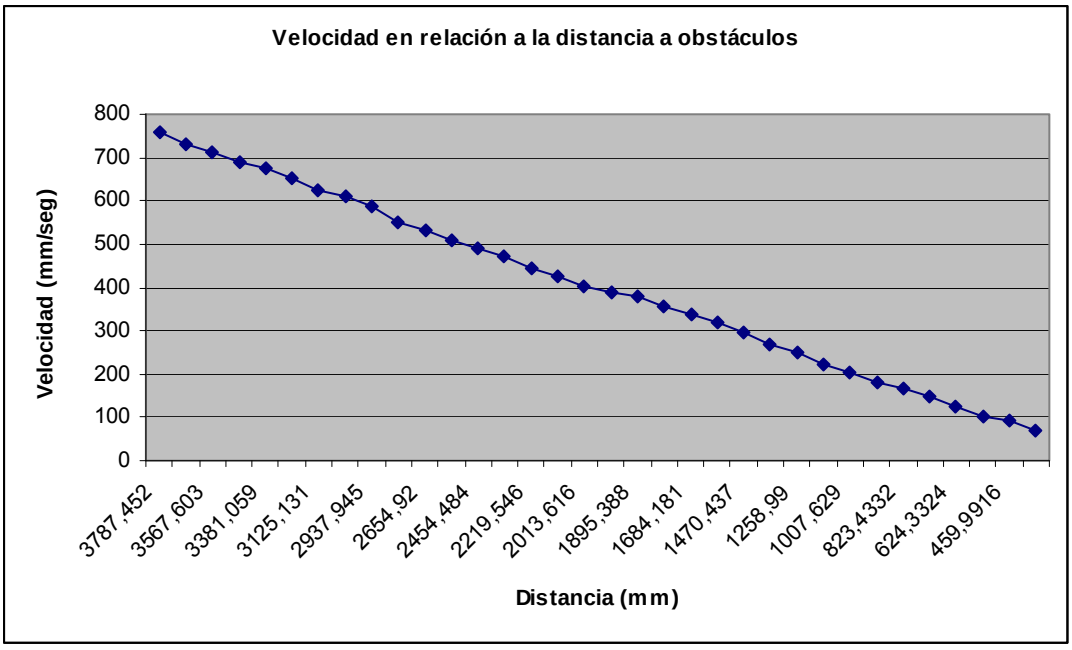

Figura 4.8: Relación distancia al obstáculo y la velocidad del robot en el simulador

Como puede observarse en la gráfica de la figura 4.8, este comportamiento funciona satisfactoriamente en el simulador, cuanto más cercano se encuentra el robot al obstáculo, menor es la velocidad del mismo.

4.2.3.3.Pruebas en entorno real

Al igual que sucedía en las pruebas realizadas en el simulador, tan solo se comentan los resultados obtenidos con el árbol de decisión definitivo utilizado para la predicción de choques. El resto de pruebas realizadas con otros árboles quedarán detalladas en la sección 4.3 del presente documento.

Al probar este árbol de clasificación en un entorno real se consiguen resultados satisfactorios. Los resultados concretos que se obtuvieron fueron de 11 choques de un total de 282 situaciones de choque, lo que supone un 3,90% de fallo, o lo que es lo mismo un 96,10% de fiabilidad, siendo este un valor aceptable.

Por otro lado, el control de motores en el entorno real presenta unos resultados que se pueden apreciar en la gráfica de la figura 4.9.

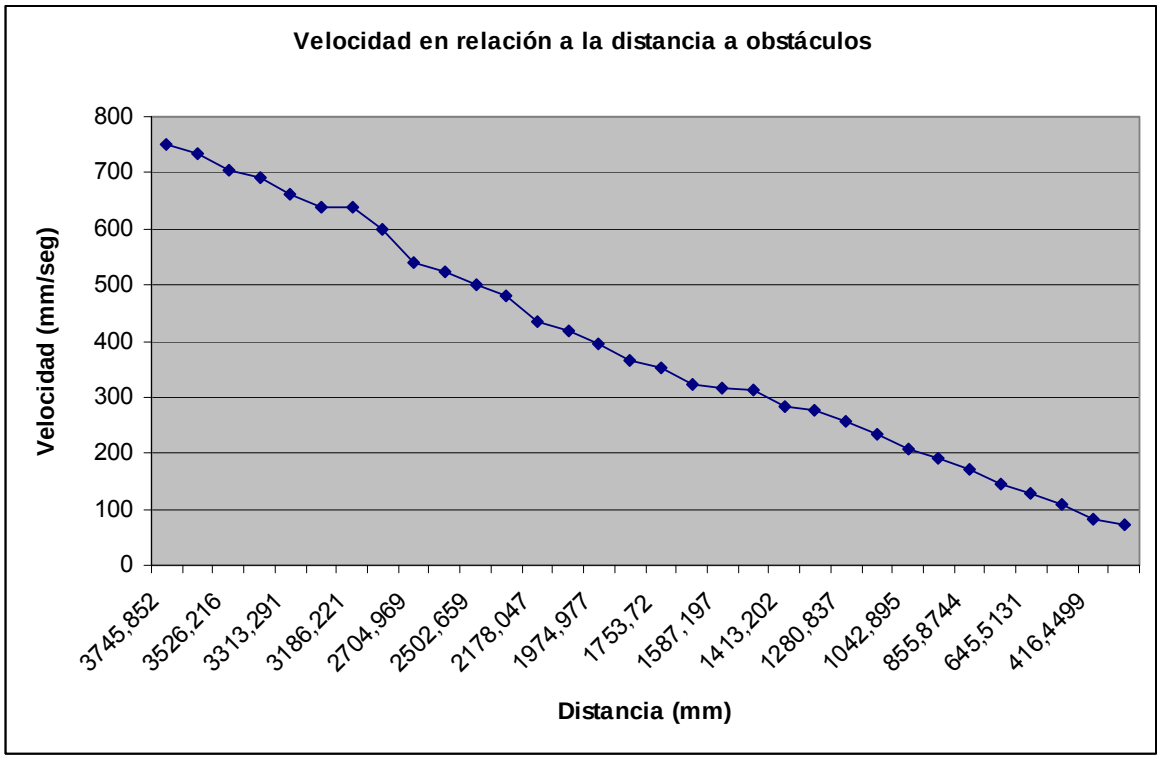

Figura 4.9: Relación distancia al obstáculo y la velocidad del robot en un entorno real

Estas pruebas también dan un resultado correcto ya que la velocidad impuesta al robot disminuye a medida que la distancia a un obstáculo también es menor.

4.2.3.4.Pseudocódigo

```
Evitar{
      lecturas_sonares=obtenerLecturaSonares();
      choca = sentirObstaculo();
      si(choca){
            velocidad = 0;
            si (chocaLateralIzquierdo(lecturas_sonares)){
                  cambiarOrientacion(orientación+80);
            }
            si no{
                  cambiarOrientacion(orientación-80);
            }
      }
      Si no{
            vagar();
      }
}

SentirObstaculo{
      lecturas_sonares=obtenerLecturaSonares();
      return validarArbolDecision(lecturas_sonares);
}

Vagar{
      lecturas_sonares=obtenerLecturaSonares();
      distancia_obstaculo=obtenerDistanciaMinima(lecturas_sonares);
      establecerVelocidad(distancia_obstaculo*0.51);

}
```

Tal y como se observa en el pseudocódigo si se detecta un obstáculo con el cual el robot podría chocar, se modifica la orientación del robot a izquierda o derecha según la posición del obstáculo.

En el caso de que no se detecte obstáculo con el que chocar, el robot continúa el movimiento por el entorno sin variar su orientación, pero con una velocidad proporcional a la distancia a la que se encuentra el obstáculo más cercano.

4.2.4. Nivel 2

Hasta ahora se ha visto el desarrollo de lo que se podrían llamar acciones de supervivencia del robot, es decir las acciones que el robot debería ejecutar para evitar lo que para él es fundamental: no chocar e incluso poder desplazarse sin chocar con los obstáculos estáticos que encuentra a su paso.

Una vez llegados a este punto, se tienen los dos niveles iniciales de la arquitectura de subsunción, por lo que ahora procede a completar aun más la ya mencionada arquitectura con su nivel 2, en el que se dota al robot de la capacidad de “explorar” el mundo, de ver lugares en la distancia y utilizando el conjunto de sus sensores de los que se dispone. Se debe recordar que los robots con los que se está trabajando, no están

dotados de cámara, por lo cual no podrá ver su entorno o reconocer objetos ya sean cercanos o lejanos, sino que tan sólo dispone de sus parachoques para detectar choque y sónares para detectar obstáculos y objetos a una cierta distancia.

La funcionalidad que adopta este nivel de la arquitectura es la de seguir un pasillo pero intentando buscar el centro del mismo y manteniéndose en él. Dentro del pasillo se puede encontrar un hueco, como por ejemplo unas escaleras, en el pasillo, por lo que el robot deberá despreciar estos huecos para evitar posibles caídas por él.

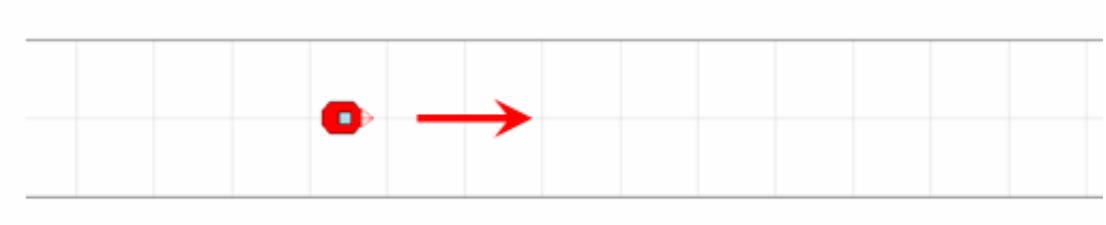

Figura 4.10: Nivel 2: recorrer pasillo

4.2.4.1.Comportamientos

En este nivel se encuentran tres comportamientos asociados a él: “mirar pasillo”, “detectar escalera” y “mantener medio”

- **Mirar pasillo**

Para conseguir centrar el robot en el centro del pasillo, se obtienen lecturas de los sónares 0 y 7 del robot, los cuales se corresponden con los sónares que toman lecturas de los laterales izquierdo y derecho del robot respectivamente.

Con la lectura de dichos sónares se obtendrá si el robot se encuentra o no centrado en el pasillo, para ello se realiza la diferencia de ambas lecturas:

```
Diferencia = sonar_0 – sonar_7
```

Se realiza la diferencia del sónar 0 menos el sónar 7, ya que de este modo si el robot no se encuentra centrado en el pasillo, es decir la diferencia no es 0, se obtiene el sentido de giro del robot, si la diferencia es negativa el robot deberá girar a la derecha ya que la lectura es mayor por tanto la distancia al lateral derecho es mayor que al lateral izquierdo, y de un modo equivalente pero en sentido contrario sucederá si la diferencia fuera positiva.

- **Mantener medio**

Este comportamiento recibirá la información procesada por el comportamiento “mirar pasillo”. En esta información tratada, si la diferencia de la distancia a ambos lados

del pasillo es distinta de 0, el robot no estará centrado en el pasillo tal y como ya se ha dicho anteriormente, por lo que el robot deberá girar para ir aproximándose al centro a medida que avanza. Pero, ¿cuánto girar? Ante esta situación hay dos posibilidades:

- La primera opción consiste en girar el robot en gran medida para que se acerque cuanto antes al centro del pasillo, y una vez en dicho centro siga avanzando a través del pasillo centrado en él. Se corresponde con la figura 4.11.

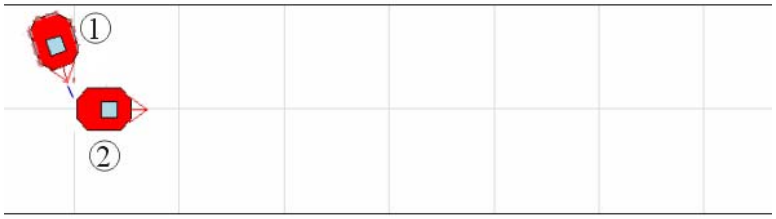

Figura 4.11: Primera opción para centrarse en el pasillo

- La segunda opción es hacer girar poco a poco el robot hacia el centro del pasillo, pero dándole prioridad al avance del pasillo, es decir el robot avanza a lo largo del pasillo acercándose cada vez más al centro del mismo y una vez allí seguir avanzando pero ya en línea recta manteniéndose centrado. Se corresponde con la figura 4.12.

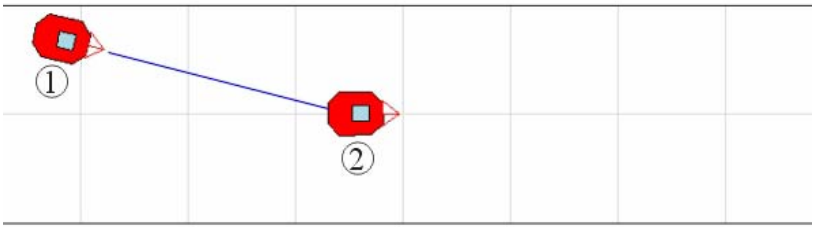

Figura 4.12: Segunda opción para centrarse en el pasillo

La alternativa escogida fue la segunda, ya que lo primordial en este comportamiento es avanzar por el pasillo, pero sin dejar de lado el centrado en el mismo.

Para obtener el ángulo de giro que debiera adoptar el robot para rectificar su posición se utiliza la distancia al centro del pasillo, o lo que es lo mismo, la diferencia entre la lectura de los dos sónares, 0 y 7. Pero para conseguir un giro más suave, se atenúa dicha diferencia mediante la función:

```
giro = abs(diferencia)/50
```

Donde *abs(diferencia)* es el valor absoluto de la diferencia.

Con esta función se consigue una distribución lineal, proporcional a la distancia a los laterales del pasillo, pero con una atenuación para suavizar el giro. Se puede apreciar la distribución de esta función en la figura 4.13.

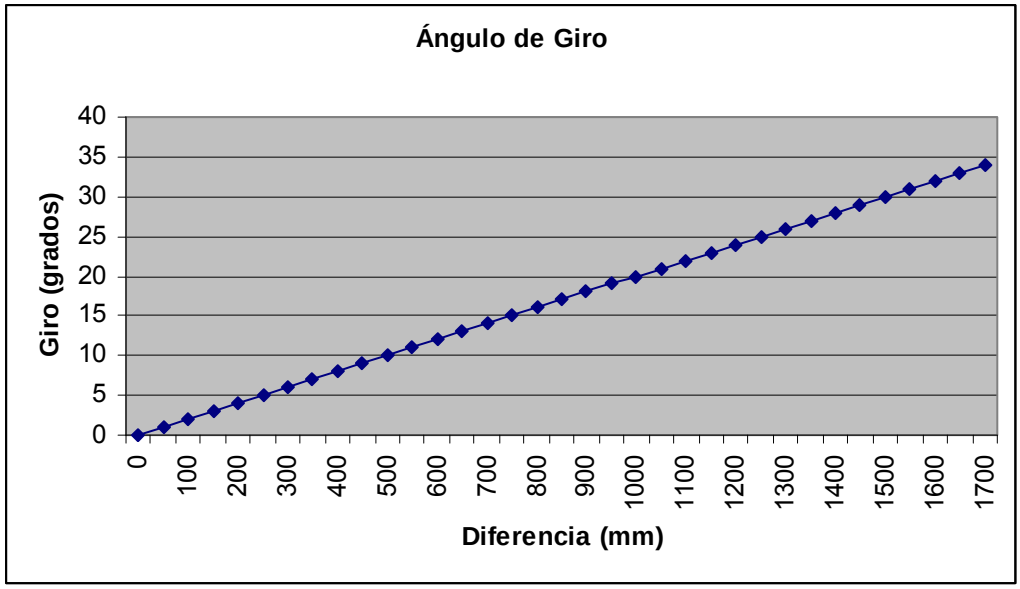

Figura 4.13: Distribución obtener centro pasillo

Con esto se consigue que con diferencias pequeñas el ángulo de giro sea pequeño, y con diferencias grandes, el ángulo de giro sea mayor. Pero sólo con esto no se consigue llevar a cabo nuestra alternativa, avanzar por el pasillo aproximándose poco a poco al centro del mismo. Para ello es necesario limitar el ángulo de giro. Se decidió establecer un valor máximo de 25 grados a girar, para así evitar la realización de un giro brusco hacia el centro. Con esta modificación la función de giro presentará la distribución que se muestra en la figura 4.14.

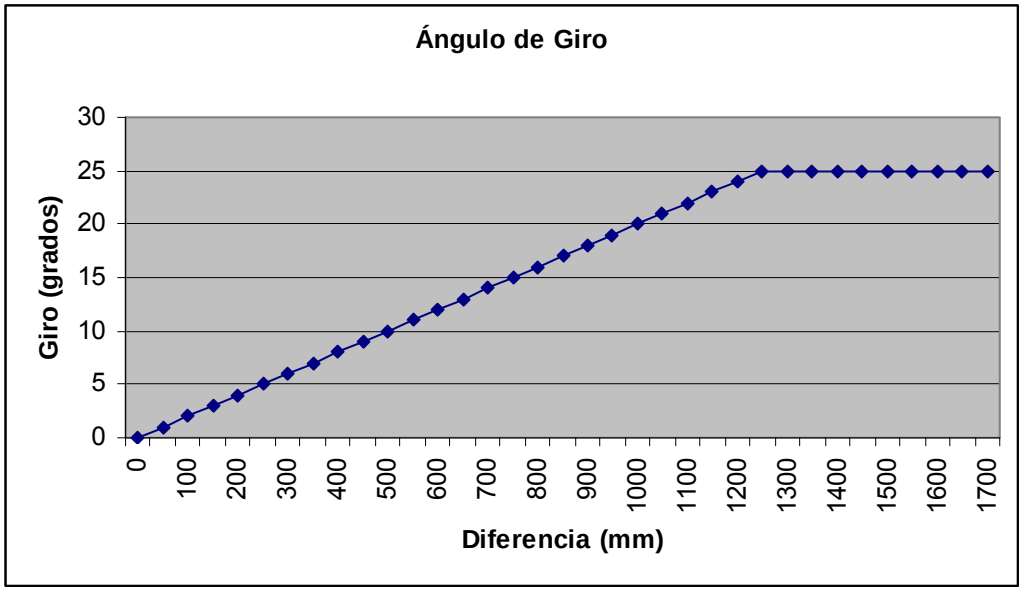

Figura 4.14: Distribución obtener centro pasillo con limitación

Una alternativa a esta función podría ser utilizar una función exponencial de tal modo que el comportamiento fuera similar, con valores grandes cuando la diferencia es grande, y valores cercanos a cero en el caso contrario. El problema que se encuentra con estas funciones es que en situaciones cercanas a cero se atenúa demasiado y el tiempo que tarda en centrarse es notablemente mayor.

Al validar el funcionamiento de este comportamiento se apreció que en el caso de que el robot, en su punto de partida, no posea una orientación paralela al pasillo, no llega a centrarse totalmente, sino que se estabiliza más próximo a un lateral del pasillo que al otro. Para conseguir solventar esta situación, se estableció una variable que contabiliza las interacciones en las que la diferencia entre la lectura de los sónares 0 y 7 permanecen constantes, es decir está estable. Si esta situación de estabilidad se repite durante 10 iteraciones, y no se está en el centro, o lo que es lo mismo el giro es distinto de 0, se deberá realizar una modificación de la posición, girando hacia el centro, pero ya que se está muy cercano a él (como se ha dicho anteriormente se está ante una situación de estabilidad próxima al centro) se deberán realizar movimientos muy suaves para que no se produzcan giros abruptos del robot, por ello se aplica un giro que sigue la siguiente función:

```
Modificacion = abs(diferencia)/100
```

Con este mecanismo, además se consigue suavizar la estabilización en el centro del pasillo.

A esta altura del desarrollo del comportamiento, se consigue que el robot se recorra el pasillo centrado en él. Pero ¿Qué sucede si encuentra un obstáculo en su camino?

Para contestar esta pregunta, se debe recordar que debajo de este nivel de la arquitectura, se tienen otros niveles que hacen que el robot se mueva detectando obstáculos y posibles choques con estos, en el caso de que se pudiera producir un choque, se le indicaba un giro en la orientación del robot.

Una vez recordado esto, ya se puede intuir cuál sería el comportamiento del robot. Este será el de avanzar por el pasillo y al detectar el obstáculo con el que chocar, la arquitectura provocará que el robot evite el obstáculo y siga recorriendo el pasillo.

Para explicar cómo se resuelve esta situación, y que la explicación sea lo más clara posible, se exponen en la figura 4.15, las diferentes fases por la que pasa para solventar la situación.

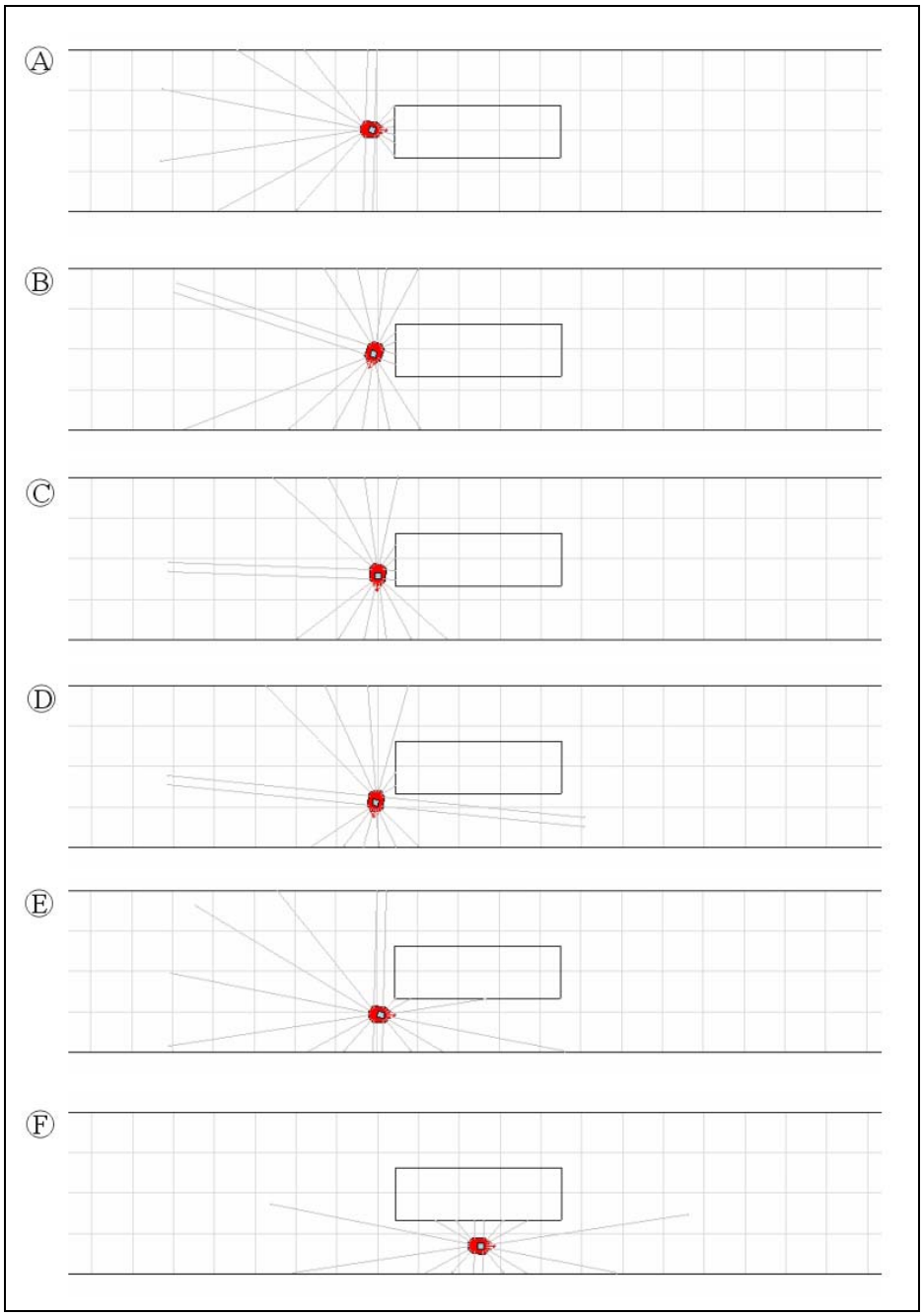

Figura 4.15: Pasos para evitar obstáculo

1. Al **detectar el obstáculo** y determinar por el nivel 1 que se produciría un choque. Este hecho se representa en la figura 4.15-A.

2. El nivel 1 de la arquitectura, al detectar el obstáculo provoca un **giro del robot** para evitarlo.

3. **Esperar hasta que el robot gira**. Al girar, el sónar derecho o izquierdo del robot (0 o 7) dependiendo del sentido del giro. Este hecho se representa en la figura 4.15-B.

4. Según el comportamiento "evitar"**, el robot avanza recto** comprobando la presencia del obstáculo por sus sónares. El robot se mantiene avanzando hasta que el sónar que toma la referencia al obstáculo no detecte al mismo. Esta situación se muestra en las figuras 4.15-C y 4.15-D.

5. Cuando deja de detectarse el obstáculo el robot irá **recuperando la orientación que tenía en el momento anterior a evitar el obstáculo**. Representado en la figura 4.15-E.

6. Con la orientación recuperada, el robot continua con recorriendo el pasillo centrándose en él, ya sea el pseudo-pasillo que se crea entre el obstáculo y la pared del pasillo real, o en el pasillo real si el obstáculo está totalmente superado. Se encuentra representado en la figura 4.15-F.

Una vez que el robot supera el obstáculo, vuelve a establecer la orientación inicial que poseía el robot antes de comenzar a evitar el obstáculo.

- **Detectar escalera**

A lo largo de un pasillo se pueden encontrar unos huecos tales como unas escaleras. Por tanto, es necesario tener en cuenta estas situaciones para recorrerlos sin caer por estos huecos.

Para conseguir evitar que el robot se mueva hacia el hueco, se realizan dos hechos. El primero de ellos es el de no permitir al robot distanciarse de la pared opuesta a las escaleras a una distancia mayor a 2 metros. En el caso de que el robot intente distanciarse más, se modificará su orientación para que vuelva a acercarse. Con esto, adicionalmente se consigue que no se pierda la referencia a la pared, y por tanto en el caso del hueco de la escalera evitar que si se separa de la pared existente y que en la otra no haya, el nivel 3 no piense que se está ante una esquina.

En segundo lugar, se incluyó otra condición de seguridad para despreciar el hueco de la escalera. Para ello el robot seguirá recto sin modificar su situación en el pasillo para centrarse siempre y cuando la lectura del sónar situado en el lateral de la escalera obtenga una lectura igual o superior a 5000 (hay un hueco porque no obtiene lectura) o la diferencia con la lectura anterior sea muy elevada, mas de un metro de diferencia y por tanto está ante un hueco (siempre y cuando la lectura anterior no fuera cero). Esto sólo se tendrá en cuenta si el robot no está evitando ningún obstáculo, ya que en el caso de estar evitándolo y se ejecutará esta condición, el robot sería incapaz de recuperar su orientación al esquivar el obstáculo.

4.2.4.2.Pruebas en el simulador

Para validar este comportamiento se creó en el simulador un pasillo, y se probó que el robot se centrara en el pasillo y se moviera a lo largo de él de manera estable. El primer pasillo en el que se probó fue un pasillo vacío, es decir sin obstáculos en él.

En la primera prueba el robot partía desde una posición centrada en el mismo (punto 1 de la figura 4.16) y se comprueba como el robot se mueve a lo largo del pasillo centrado en él (hasta punto 2 de la figura 4.16), tal y como se muestra en la figura 4.16.

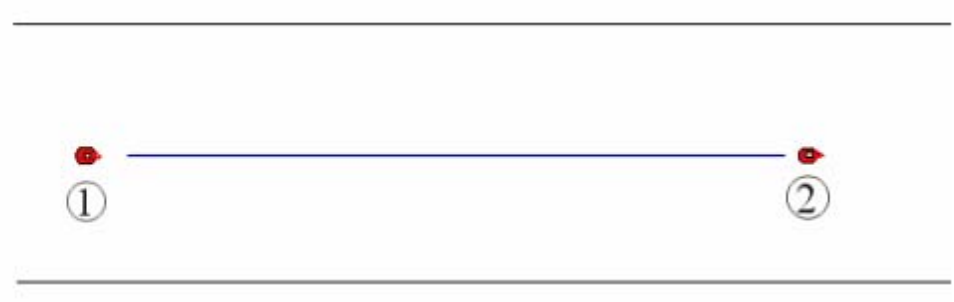

Figura 4.16: Primera prueba recorrer pasillo.

En la siguiente prueba se cambió el punto de partida, y en esta ocasión el robot partía de una posición cercana a una de las paredes del pasillo (punto 1 de la figura 4.17). El comportamiento del robot en esta ocasión es rectificar su posición aproximándose poco a poco al centro del pasillo, y una vez centrado en él (punto 2 de la figura 4.17) mantenerse centrado mientras sigue recorriendo el pasillo (hasta el punto 3 de la figura 4.17).

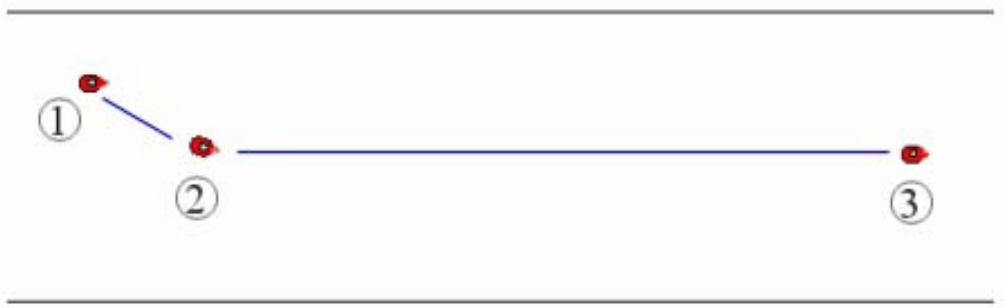

Figura 4.17: Segunda prueba recorrer pasillo.

Para validar el funcionamiento de la arquitectura de subsunción se incluyeron en los pasillos obstáculos para que el robot lo esquivara mediante el uso de los niveles inferiores de la arquitectura.

Una prueba para validar éste hecho fue la que se muestra en la figura 4.18

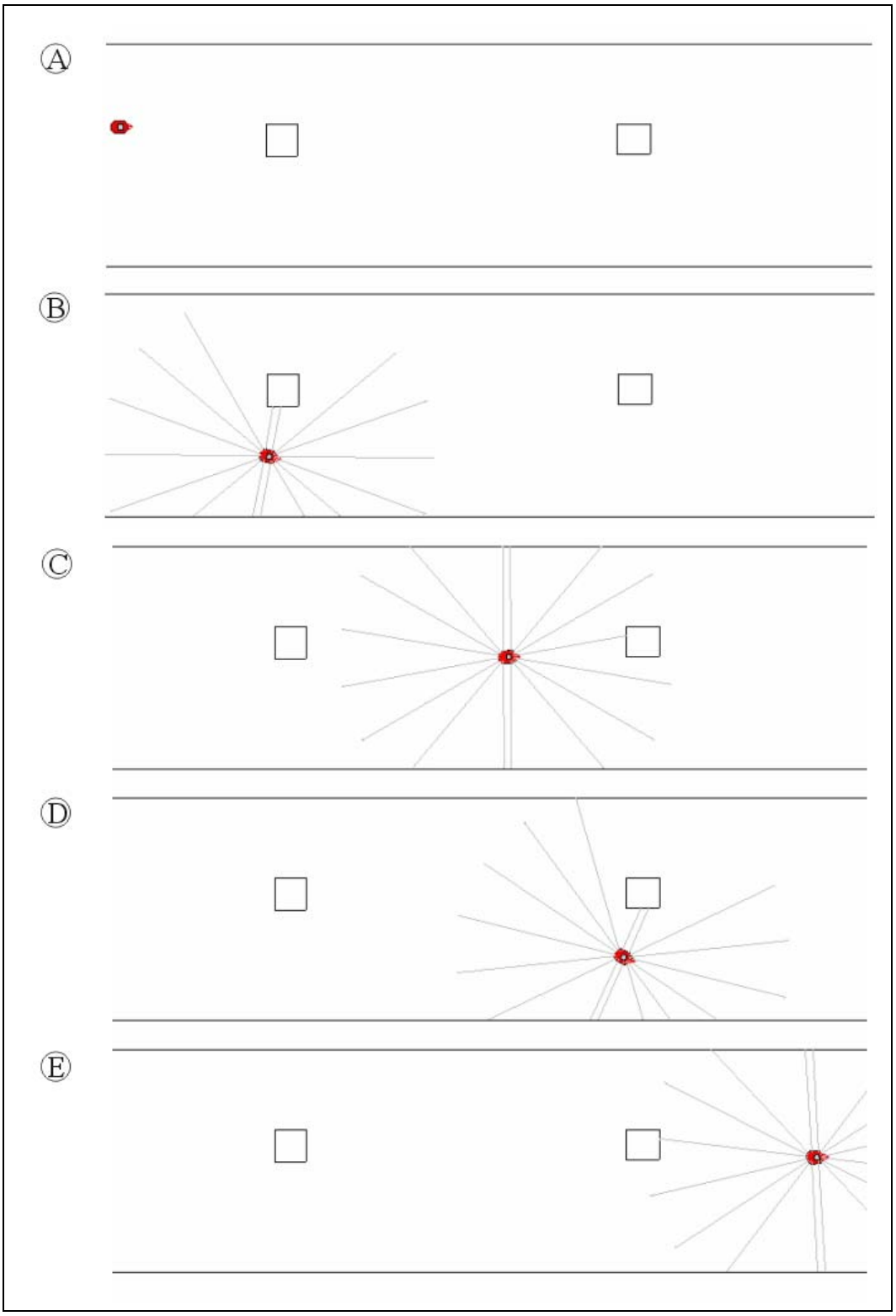

Figura 4.18: Tercera prueba recorrer pasillo. Con obstáculos

En esta situación el robot comenzaba centrándose en el pasillo (figura 4.18-A), al conseguirlo encontraba un obstáculo a su paso. Al encontrar este primer obstáculo lo esquivaba (figura 4.18-B). Una vez esquivado el obstáculo el robot de nuevo se centra en el pasillo por el que se mueve (figura 4.18-C). Sigue el mismo movimiento y vuelve a

encontrarse otro obstáculo y vuelve a operar del mismo modo, esquiva el obstáculo (figura 4.18-D) y se centra de nuevo siguiendo su camino (figura 4. 18-E).

Como se aprecia por esta prueba, el funcionamiento es correcto y la subsunción funciona correctamente subsumiendo los giros del robot para evitar los obstáculos y seguir recorriendo el pasillo.

Se realizaron pruebas similares a la detallada en un total de siete escenarios distintos tal y como se muestran en la figura 4.19.

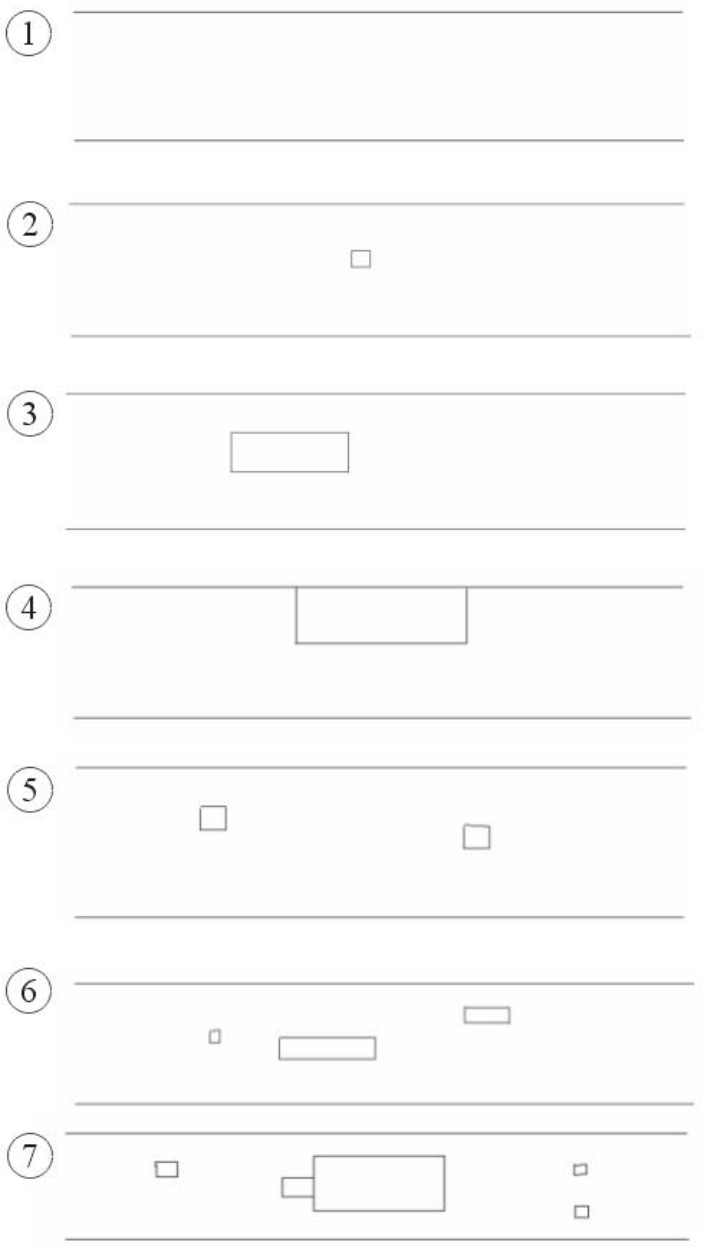

Figura 4.19: Modelos de pasillos de pruebas

Se realizaron un total de 68 pruebas para recorrer los pasillos. Respecto a la situación de partida del robot en el pasillo es variado: centrado en un extremo del pasillo,

en un lateral del extremo del pasillo, cercano a algún obstáculo y paralelo y no paralelo al pasillo.

De las 68 pruebas realizadas, la ejecución falló en 2 ocasiones debido a un choque con el obstáculo cuando lo estaba esquivando el robot. Estos choques se pueden ver en la figura 4.20.

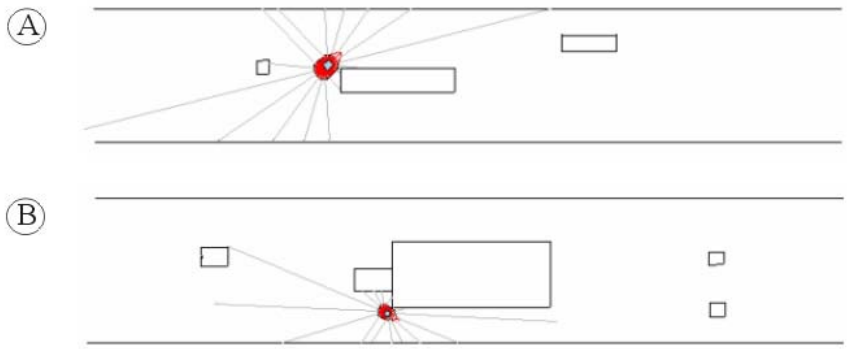

Figura 4.20: Situaciones de choque al recorrer pasillo

El caso del choque mostrado en la figura 4.20-A el robot partía desde el extremo izquierdo del pasillo y centrado en él. El robot esquiva el primer obstáculo que encuentra, rectifica su posición e intenta esquivar el segundo, pero intenta retomar la orientación por la que debe recorrer el pasillo antes de lo debido y choca con la esquina del obstáculo. Hay que destacar que esta situación se ve agravada en el simulador debido a que no dispone de dispositivos parachoques por lo cual el robot no puede recuperarse ante el choque, sin embargo en un entorno real el robot será capaz de reponerse ante esta situación.

El segundo choque, mostrado en la figura 4.20-B, el robot partía situado frente a la pared de la esquina con la que choca, el comportamiento del robot en este caso es intentar centrarse pero al empezar a superar la esquina en una situación que parecía posible, al producirse un pequeño giro la parte media del robot choca con la esquina.

Tras superar satisfactoriamente estas pruebas en un 97.05%, se realizaron también pruebas en el entorno real.

4.2.4.3.Pruebas en entorno real

Hay que destacar que estas pruebas reales se realizaron en la segunda planta del edificio Sabatini del campus de Leganés de la Universidad Carlos III de Madrid. En este edificio, los pasillos no tienen las paredes laterales totalmente rectas, sino que debido a las puertas de los laboratorios y clases, el pasillo tiene diferente niveles de ancho, siendo más ancho en los lugares de las puertas; esta característica del pasillo se puede ver en la figura 4.21. Por este motivo, al robot le costaba más conseguir estabilizarse, ya que cuando lo conseguía, encontraba que el pasillo se ensanchaba debido a que encontraba una puerta con lo cual intentaba centrarse ante esta nueva situación, pero al sobrepasar la puerta, el pasillo volvía a disminuir con lo cual de nuevo debía modificar su posición.

No obstante el robot se estabilizaba y movía suavemente siempre que la situación lo permitía.

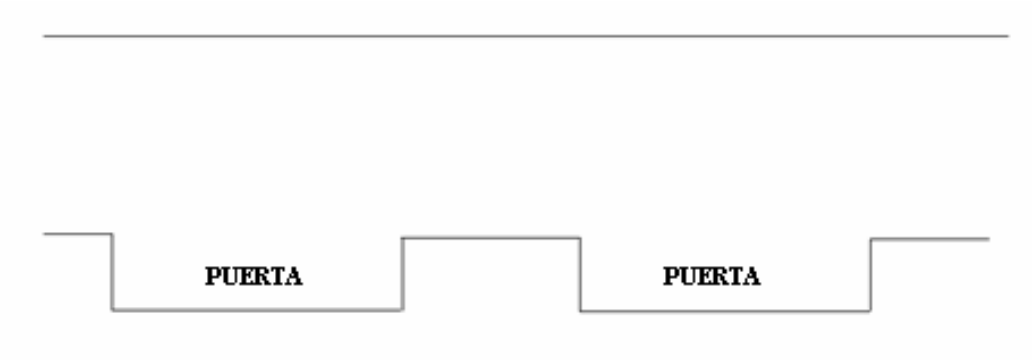

Figura 4.21: Modelo del pasillo a recorrer en el entorno real

En el entorno real, del mismo modo que en el simulador, se realizaron pruebas con diferentes distribuciones de obstáculos y finalidades: sin obstáculos en el pasillo para validar que el robot consigue centrarse en el centro del pasillo y estabilizarse en él mientras lo recorre, y al incluir los obstáculos se pretende validar la esquivación de los obstáculos y cómo se consigue mantener el recorrido del pasillo en el sentido adecuado.

Las pruebas realizadas en el pasillo sin obstáculos tuvieron un comportamiento muy similar al mostrado en las figuras 4.16 y 4.17, con la salvedad de que la anchura del pasillo no es constante con lo cual varía la posición en la que se estabiliza el robot a lo largo del pasillo.

El resto de las pruebas realizadas se realizaron con obstáculos a lo largo del pasillo. Es reseñable que los obstáculos utilizados son obstáculos estáticos para que el robot pudiera percibirlos y así no encontrarse repentinamente con un obstáculo. Al igual que para las pruebas en el simulador se realizaron distintas distribuciones de los obstáculos, así como la variación del punto de partida del robot.

Se realizaron un total de 71 pruebas de las cuales hubo tres situaciones en las que el robot chocó. Cada uno de los tres choques se produjo mientras el robot intentaba esquivar un obstáculo humano estático. El comportamiento que se producía en el robot ante el primer choque fue:

1. El robot detectaba el obstáculo a esquivar.

2. Giraba hacia la izquierda o la derecha (según convenga) del obstáculo para esquivarlo.

3. Avanzaba y dejaba de detectar el obstáculo debido a no recibir eco de la señal sónar lateral, a pesar de no haber rebasado por completo el obstáculo.

4. Intentaba recuperar la orientación para seguir recorriendo el pasillo en el sentido adecuado.

5. Se encontraba de nuevo con parte del obstáculo que no había sido rebasado por completo.

En las otras dos situaciones de choque el comportamiento del robot era:

1. El robot detectaba el obstáculo a esquivar.

2. Giraba hacia la izquierda o la derecha (según convenga) del obstáculo para esquivarlo.

3. Avanzaba y dejaba de detectar el obstáculo, en esta ocasión tras haber rebasado por completo el obstáculo.

4. El robot recupera la orientación para seguir recorriendo el pasillo en el sentido adecuado.

5. Avanza lentamente y determina que debe girar para centrarse en el pasillo a partir de la lectura de los sónares, pero al girar se encontraban con el pie del individuo que en ese momento era el obstáculo.

Es reseñable que a pesar de los pocos deseados choques, gracias a los parachoques y a los comportamientos de niveles inferiores asociados al mismo, el robot conseguía recuperarse de esta situación extrema.

4.2.4.4.Pseudocódigo

```
MantenerMedio{
      diferencia = MirarPasillo();
      si (DetectarHueco()==true){
            mantenerOrientacion();
      }
      si no{
            giro= CalcularAnguloGiro(diferencia);
            sentido_giro = obtenerSentido(diferencia);
            cantidad_girar= rectificarGiro(diferencia, giro);
            realizarGiro(cantidad_girar*sentido_giro);
      }
}

MirarPasillo{
      lecturas_sonares_laterales = lecturasSonaresLaterales();
      return diferencia(lecturas_sonares_laterales);
}

DetectarHueco{
      lecturas_sonares_laterales = lecturasSonaresLaterales();
      si (hayHueco(lecturas_sonares_laterales)==true){
            return true;
      }
      return false;
}
```

Cómo se observa en el pseudocódigo lo primero que se realiza es obtener la lectura de los sónares y determinar la posición relativa del robot en el ancho del pasillo. Una vez tomada la posición, se valida si se está ante un hueco en el lateral del pasillo o no, ya que en el caso de encontrarse en esta situación, el robot no debe rectificar su posición si no que deberá seguir en la misma.

En la situación en la que el robot deba rectificar su posición, se calculará tanto el sentido del giro como la cantidad a girar para una vez calculado realizar el giro.

4.2.5. Nivel 3

Llegado a este nivel, nuestro robot consigue detectar situaciones de choque, evitar de este modo el choque o en el caso de chocar recuperarse gracias a su detección por los parachoques. Además se consigue que sigua un pasillo sin perder su orientación.

En el presente nivel se da un paso más y lo que se realiza es recorrer una planta de un edificio, en concreto se pretende recorrer una planta de edificio que se encuentra distribuida con cuatro pasillos formando un cuadrado; las esquinas se caracterizan justamente por no existir ninguna referencia en ellas. Estas características se pueden observar en la figura 4.22.

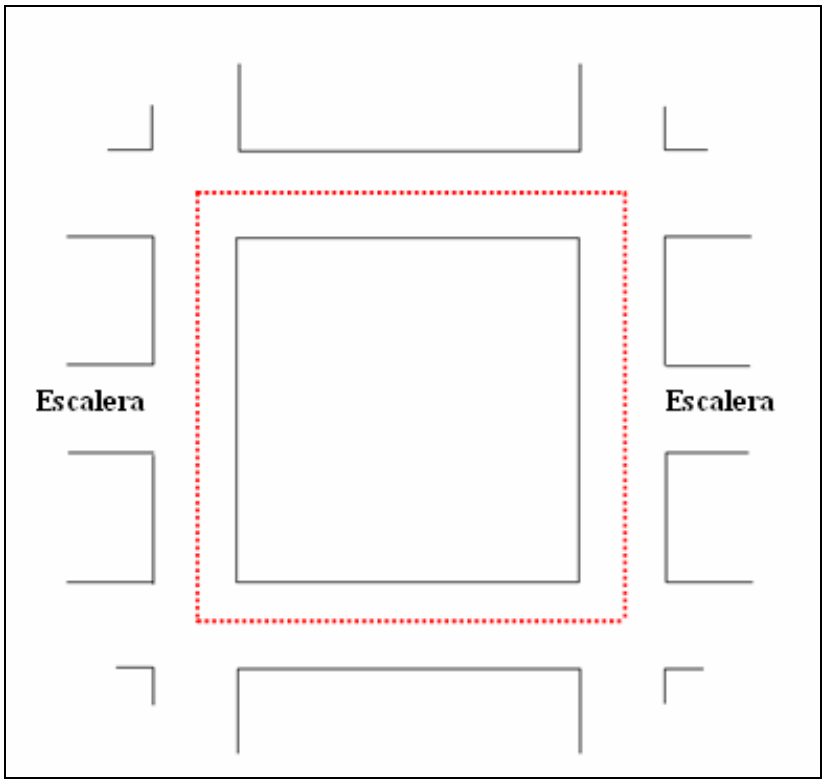

Figura 4.22: Descripción de entorno de pruebas para recorrer una planta

A este nivel, y al llevar por debajo los comportamientos asociados a recorrer un pasillo, lo necesario para este nivel es detectar las esquinas, que el robot sepa que es una esquina dentro del entorno en el que se mueve, y una vez detectada poder girar y continuar por un nuevo pasillo de los que conforman la planta.

4.2.5.1.Comportamientos

- **Detectar esquina**

Como se puede observa en la figura 4.22 del plano de la planta a recorrer, una esquina queda determinada por no tener referencia en ninguno de sus dos laterales, y que frente al robot tampoco posea obstáculo. Por ello se utilizarán los sónares laterales 0 y 7 tal y como se ha venido utilizando hasta ahora, pero adicionalmente se utilizarán los sónares 3 y 4. En la figura 4.23 se puede ver reflejado en rojo qué sónares son los que se utilizan para detectar las esquinas.

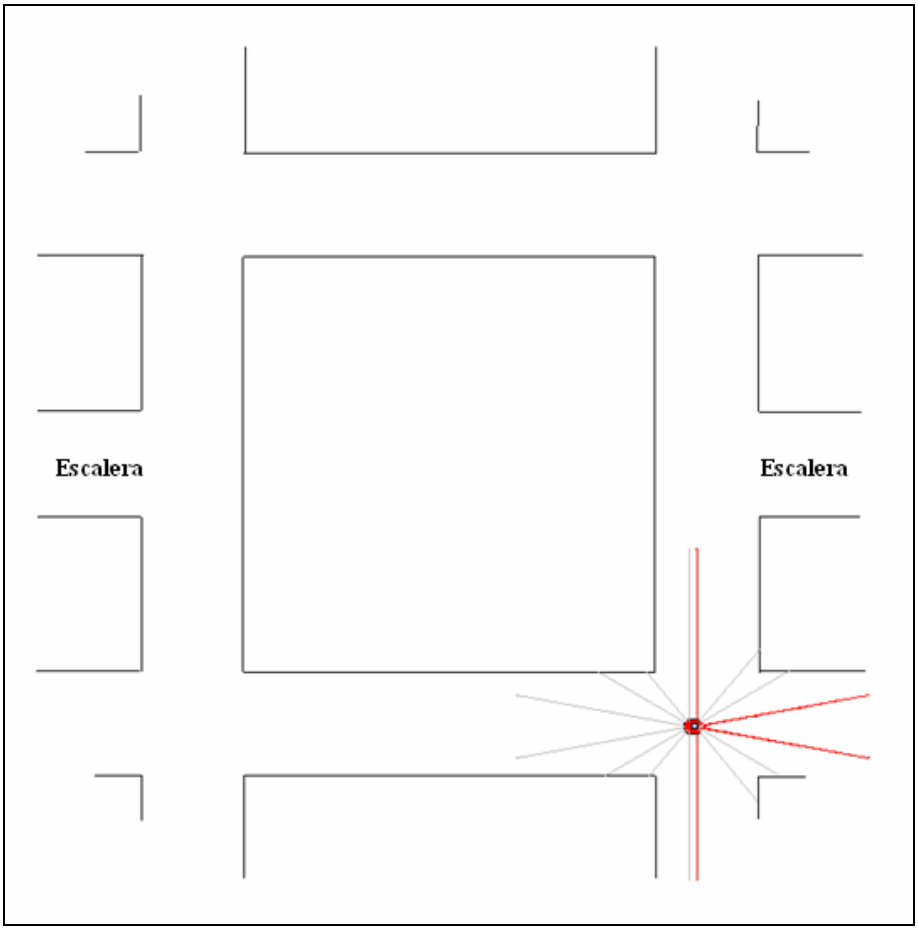

Figura 4.23: Detección de esquinas

Esta tarea parece simple, pero se han encontrado diferentes problemas que poco a poco se han ido resolviendo:

1.- El primer problema que se encontró fue el hecho de que al lanzar este comportamiento sobre el robot físico, éste detectaba en un primer instante, nada más ser lanzados los comportamientos, que estaba en una esquina sin ser esto cierto.

Esto es debido a que durante un pequeño instante de tiempo, hasta se supera el tiempo de respuesta de los sónares, el robot devuelve valores erróneos como lecturas de los mismos, valores que indican distancias mayores a las reales. Este hecho hacia que el robot, al recibir lecturas de las distancias de 5000mm determinara que estaba ante una esquina, situación falsa en la realidad.

Para conseguir solventar este problema, se introdujo un retardo en el presente comportamiento de recorrer planta. Dicho retardo consiste en despreciar en un primer momento todas las lecturas de los sónares, hasta que las lecturas parezcan ser correctas. Estas lecturas se considerarán correctas cuando cualquiera de los ocho sónares que dispone el robot, obtenga una lectura menor a los ya mencionados 5000mm; es ahora cuando se consideran las lecturas como correcta, ya que es en este instante cuando se valida que los sónares ya están recibiendo ecos de las señales emitidas, ya no recibirá el valor máximo devuelto de 5000mm salvo que realmente no se detecte nada por el *sónar*.

2.- Una vez solventado este primer problema, y el robot consigue salir correctamente recorriendo el pasillo en busca de la esquina, se detecta un segundo fallo. El problema encontrado ahora consiste en detectar de nuevo una esquina cuando en realidad no se encontraba en ella.

Dado el funcionamiento del comportamiento, el problema debería residir en la lectura de los sónares, y así era, había instantes en los que el robot no recibía lectura de los sónares por tanto el paquete recibido le daba valor de 5000mm, por tanto el comportamiento determinaba que era una esquina. Como puede observarse se presenta el mismo problema que en la situación inicial, salvo que ahora sucede en mitad de la ejecución, por lo que no se puede solucionar del mismo modo ya que si se procediera de igual manera, nunca se conseguiría detectar una esquina. Pero cabe preguntarse, ¿por qué no se reciben las lecturas de los sónares?

Pues bien, el problema es intrínseco a la tecnología utilizada: se puede producir reflexión de la señal enviada con lo cual no se recibe respuesta y por tanto no puede darse una medición correcta. Este hecho se puede ver ilustrado en la figura 4.24.

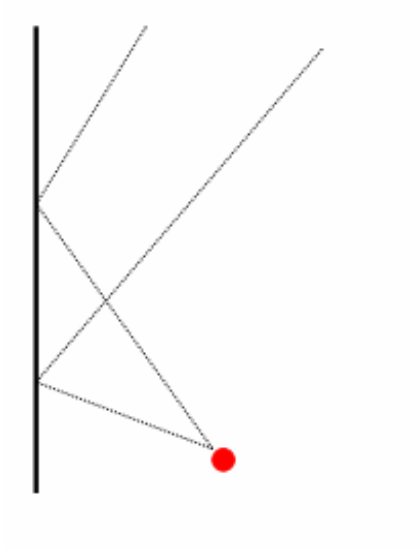

Figura 4.24: Reflexión de los sónares

Otra de las circunstancias por la cual se puede perder la señal, es por el ruido del ambiente, que puede provocar la perdida total de la señal, o que la información llegue distorsionada y si dicha distorsión es demasiado elevada, la lectura será totalmente errónea.

Ahora bien, se podría realizar una nueva pregunta, ¿Por qué no se recibe información de ninguno de los sónares implicado en la detección del pasillo? Esto se debe a que al poseer el robot los sónares muy cercanos los unos de los otros, podría darse el caso de que la señal que envía uno de ellos, lo recibiera otro y entrara en conflicto. Para evitar esto, el robot no lanza todas las señales de sus sónares de manera simultánea, sino que las lanza alternativamente. Por este motivo, puede darse la situación de no tener la lectura de ningún sónar, debido a que por reflexión o ruido en el entorno no recibe repuesta y por lo tanto no toma lectura, y ya que el robot lanza las señales alternativamente, los demás sónares involucrados en la detección de la esquina no han lanzado la señal en ese instante.

Una vez expuesto el motivo del problema, se puede explicar el modo en el que se ha solventado. Se ha aplicado lo que se ha llamado una “validación por repetición”, es decir, para validar que es una esquina, se valida la condición varias veces, un mínimo de 10, y si en todas estas comprobaciones se han cumplido las condiciones para que se esté en una esquina, entonces se toma la decisión de estar en una y se gira. En el caso de que alguna comprobación indique que no se está en una esquina, se desecha la posibilidad de girar.

Dado que hay que validar varias veces la condición de esquina, durante dicho periodo de tiempo hay que limitar la velocidad del robot, para evitar descubrir que se encuentre ante una esquina demasiado tarde, y el giro o incluso las últimas comprobaciones se produzcan demasiado tarde y ya se haya sobrepasado la esquina.

- **Tomar esquina**

Una vez que el comportamiento "detectar esquina" determina que se está ante una esquina, hay que hacer que el robot gire y tome el nuevo pasillo.

Pues bien, el robot al detectar una esquina, debe girar 90 grados, lo que provocará que el robot quede orientado hacia el nuevo pasillo.

Una vez que el robot ha girado, debe comenzar a avanzar hasta adentrarse en el nuevo pasillo que se abre ante él pero ante esta situación se encontró un problema:

- Una vez que el robot gira por estar en una esquina, debe avanzar por el nuevo pasillo que se le presenta, pero tarda un tiempo hasta que se adentra a este pasillo; durante un tiempo, tras girar el robot avanza pero sigue encontrándose en el espacio de la esquina, con lo cual en circunstancias en las que el robot gira alejado del nuevo pasillo, avanza pero detecta de nuevo esquina (la misma esquina) y vuelva a girar.

  Para solventar este nuevo problema surgido, se introduce una limitación en la detección de esquinas. Esta limitación consiste en no permitir detectar esquina durante un tiempo después de haber detectado una. Se podrá volver a detectar una esquina cuando el sónar lateral vuelva a tomar una lectura distinta a 5000mm, ya que esto querrá decir que se está dentro del nuevo pasillo, y que si detecta un pasillo, este no será el mismo del anterior ya que lo habría dejado atrás.

4.2.5.2.Pruebas en el simulador

Las pruebas se han realizado sobre el simulador *MobileSim*. Se ha creado un plano que cubre las características de la planta a recorrer, tal y como se muestra en la figura 4.22.

Se realizaron diferentes distribuciones de obstáculos a partir de los cuales realizar las pruebas, tal y como se puede observar en la figura 4.25.

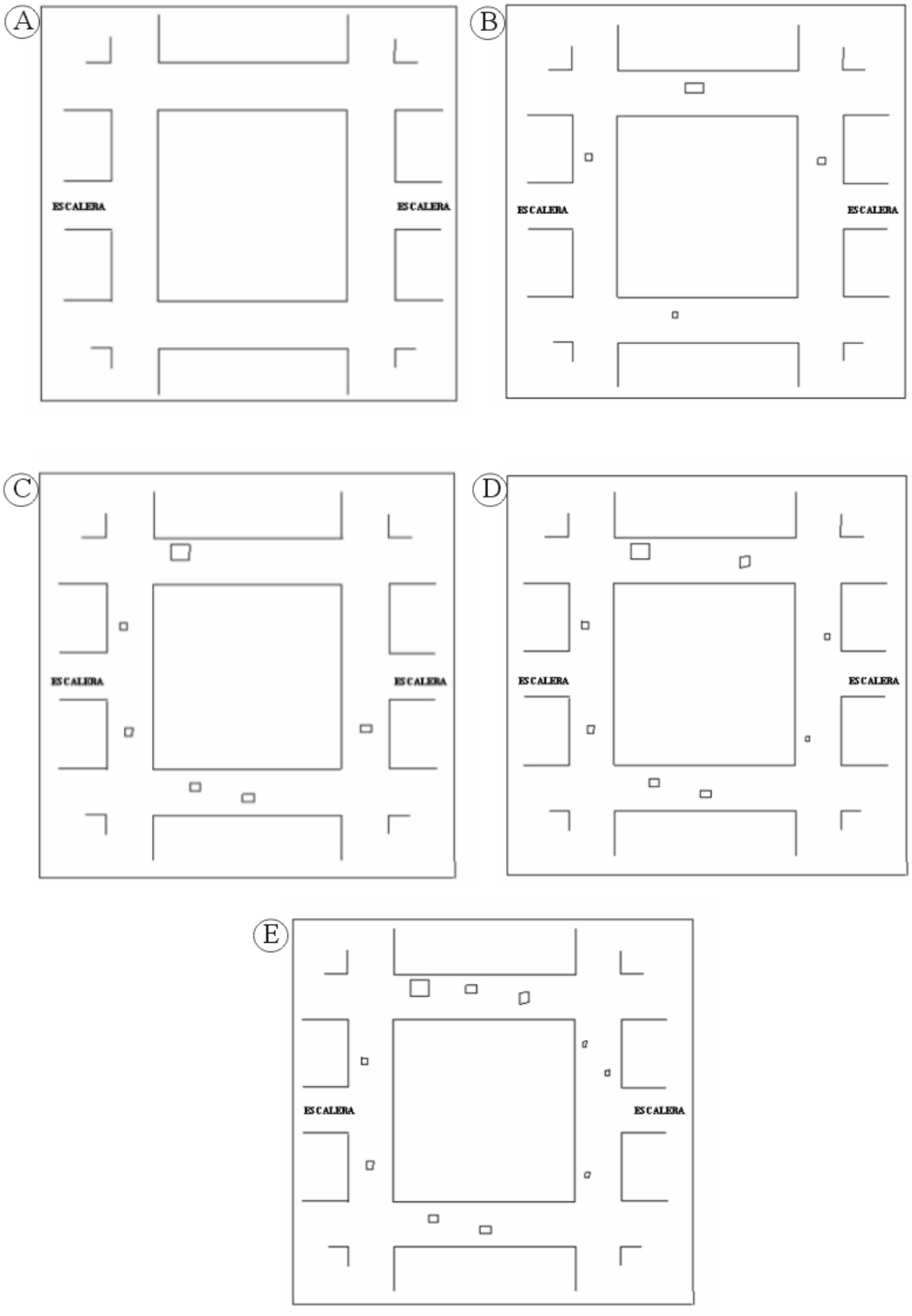

Figura 4.25: Planos de pruebas sobre las plantas

Las primeras pruebas que se realizaron en dicho simulador fueron sin incluir obstáculos en los pasillos que conforman la planta (figura 4.25-A). Y el 100% de las pruebas realizadas se superaron satisfactoriamente, el robot recorría por completo la planta del edificio detectando las esquinas cuando debían y evitando "caer" por el hueco de la escalera. En las pruebas no solo se lanzaba la ejecución de un único recorrido de la planta, sino que en una misma ejecución se recorría una media de 2 veces la planta modelada.

En pruebas sucesivas, se fueron incluyendo obstáculos en el modelo de manera que se aumentaba progresivamente la complejidad a la hora de recorrer la planta. Estos modelos que se utilizaron en las pruebas se corresponden con las figuras 4.25-B, 4.25-C, 4.25-D, 4.25-E.

Ante esta situación el comportamiento sigue detectando correctamente las situaciones de esquina, no confundiéndolas con la esquivación de obstáculos y los huecos de las escaleras seguían siendo evitados correctamente.

Sin embargo, sí que se produjeron 3 choques del total de 82 pruebas realizadas; estos choques se dieron mientras el robot intentaba esquivar algún obstáculo. Esta situación de fallo a que el árbol de decisión que se utiliza en los niveles inferiores de la arquitectura para predecir los choques no aporta una fiabilidad del 100%. Las situaciones de choque se corresponden con las mostradas en la figura 4.26.

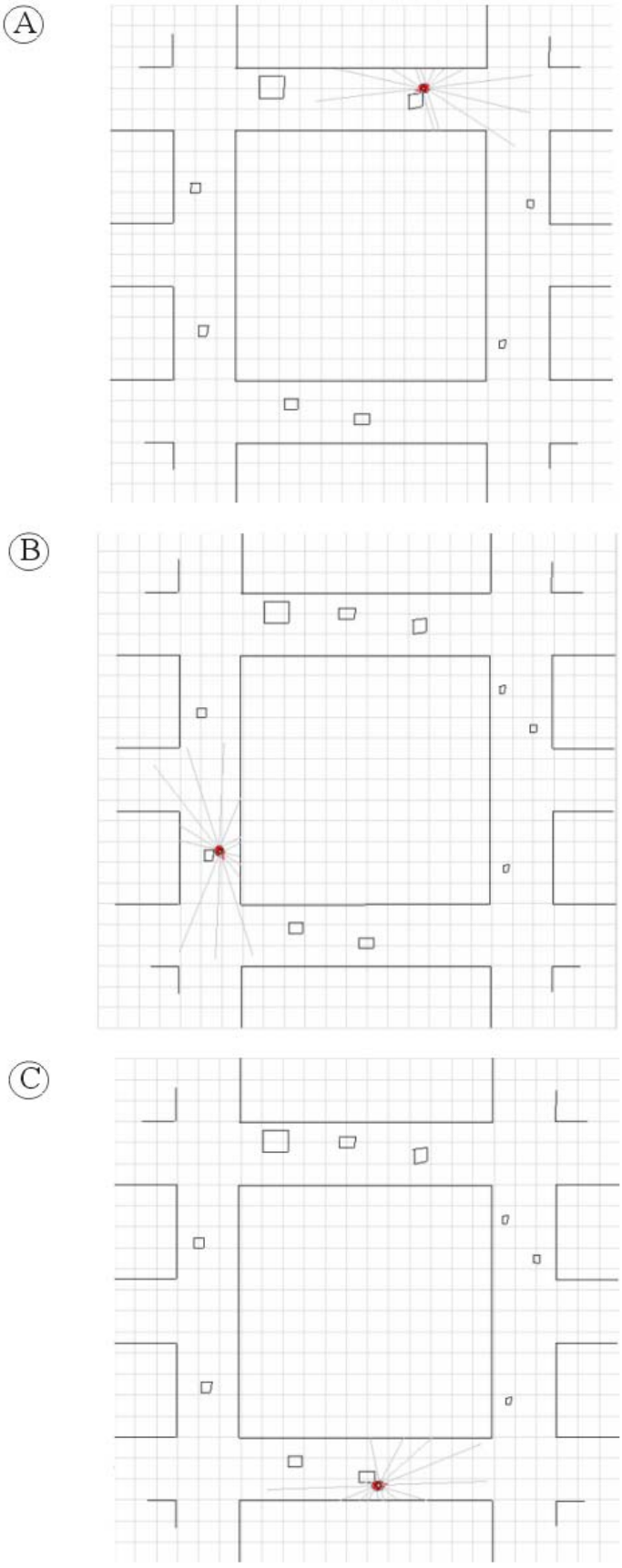

Figura 4.26: Choque al recorrer planta

En el caso del choque de la figura 4.26-C se producen cuando casi está sobrepasado el obstáculo debido a un giro por intentar centrarse ante el nuevo ancho de pasillo que se le abre.

Una situación parecida sucede con el choque de la figura 4.26-A, en el cual el robot intenta rectificar su centralidad en el pasillo y al estar situado muy cerca del obstáculo el robot choca con la esquina del mismo.

El último choque, el mostrado en la figura 4.26-B, el robot cree haber superado el obstáculo e intenta girar antes de tiempo lo que provoca que la parte trasera del robot chocara con la esquina.

4.2.5.3.Pruebas en entorno real

Una vez probado este comportamiento en un entorno simulado, se procedió a probarlo en el entorno real.

Se procedió del mismo modo que para la validación en el simulador; primero se realizaron pruebas para recorrer la planta sin obstáculos. Ante esta situación el robot detectó correctamente las esquinas y procediendo a girar y recorrer el nuevo pasillo que se abre ante él. No se dieron situaciones de detección de esquinas incorrectas y para el caso concreto del hueco del pasillo que se encuentra para las escaleras, el robot desprecia dicho hueco evitando una posible caída por él.

Superada la prueba de recorrer una planta sin obstáculos, se procedió a realizar pruebas con obstáculos estáticos situados en la trayectoria del robot. Ante ésta nueva situación el comportamiento esperado era el mismo que ante el simulador, que recorriera la planta pero se podía encontrar con alguna situación indeseada de choque, que debería ser recuperada gracias a la acción del comportamiento asociado a los parachoques; y así fue, se produjeron 4 choques ante 75 pruebas realizadas.

Adicionalmente, se puede encontrar con la situación de que en 2 de las ejecuciones realizadas el robot determinó que estaba en una esquina cuando no era así, este hecho se correspondió con evitar un obstáculo y la localización de una puerta en el extremo opuesto, tal y como se representa esquemáticamente en la figura 4.27.

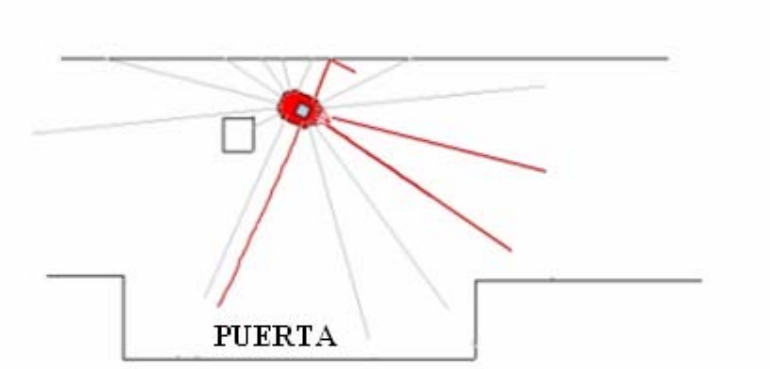

Figura 4.27: Situación de detección de falsa esquina

4.2.5.4.Pseudocódigo

```
TomarEsquina{
      Si(detectarEsquina()==true){
            validacionEsquina++;
            Si (validacionEsquina>=15){
                  Girar(orientación+90);
                  EsperarFinalizarGiro();
                  validacionEsquina=0;
            }
      }

}

DetectarEsquina{
      lectura_sonares=obtenerLecturaSonares();
      si(esLecturaValida(lectura_sonares)==true){
            return true;
      }si no{
            return false;
      }
}
```

Se puede apreciar en el pseudocódigo cómo antes de determinar si se está ante una esquina se valida que las lecturas que está tomando el robot de sus sónares son válidas o no. Cuándo estas lecturas son válidas, se comprueba si la situación que está detectando el robot se corresponde con una esquina, la comprobación deberá ser verdadera durante 15 veces consecutivas para determinar que se está ante una esquina, en cuyo caso se producirá el giro del robot y la espera para no volver a validar esquinas hasta que el robot se adentre en el nuevo pasillo que se abre ante él.

4.3. Generación de comportamiento reactivo mediante aprendizaje automático

Para conseguir obtener las situaciones en las que el robot chocaría tan sólo se puede hacer uso de la lectura que proporcionan los sensores sónares. Pero obtener estas situaciones es bastante complicado a simple vista, por ello se ha aplicado aprendizaje automático y conseguir de esta manera obtener esta información a partir de la propia experiencia del robot mediante un clasificador, en concreto un árbol de decisión.

Para obtener dicho árbol de decisión se siguieron las fases de:

1. Recopilación y preparación de datos.
2. Determinar qué tipo de aprendizaje es el más apropiado.
3. Elegir el tipo de modelo a generar (árboles de decisión).
4. Determinar el algoritmo de minería a aplicar.
5. Evaluar e interpretar los resultados obtenidos.

Para realizar la recopilación de datos se desarrolló un comportamiento por el cual el robot se desplazaba y mostraba por cada interacción la lectura que devolvían los sónares, la indicación de si está chocando o no el robot y el instante de tiempo al que corresponden los datos.

Los datos se obtuvieron de lanzar el comportamiento en el simulador *MobileSim* y obtener unos 33000 registros aproximadamente. Para obtener los registros se utilizaron diferentes entornos, es decir se lanzó el comportamiento en el simulador con diferentes mapas para intentar conseguir distintas situaciones en las que el robot detecte choques. Los distintos mapas utilizados han sido los mostrados en la figura 4.28.

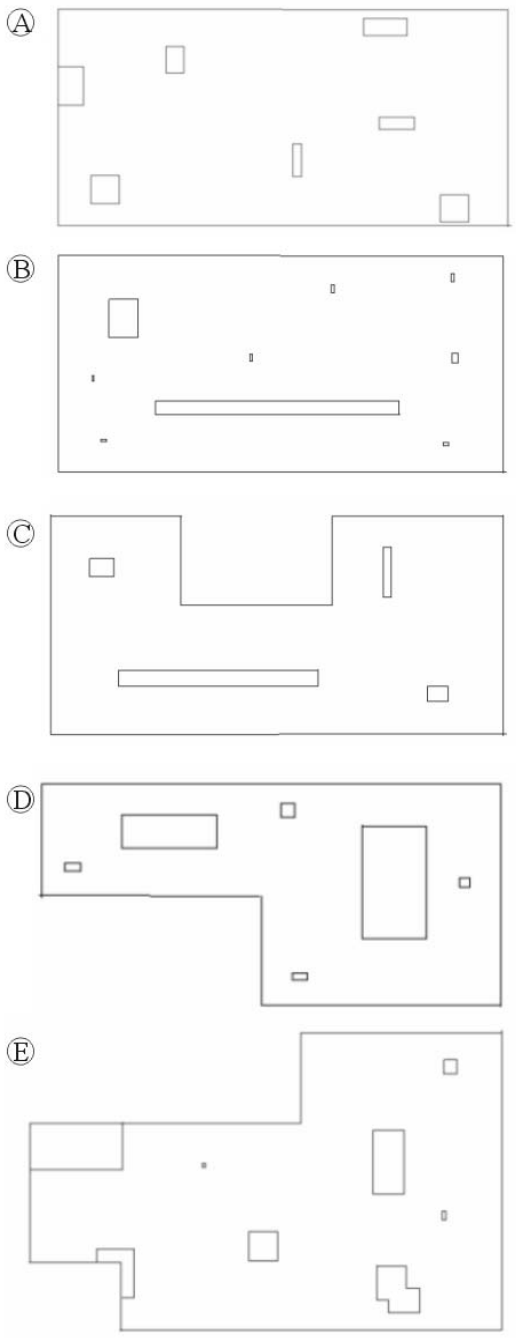

Figura 4.28: Mapas para la obtención de datos

Como puede observarse, los mapas presentan una estructura y distribución de los obstáculos distinta, el entorno cercano del robot con el cual podría chocar es distinto obteniendo de esta manera una colección de datos heterogénea.

Una vez que se obtienen los datos, es necesario preprocesarlo, ya que el objetivo no es decidir cuando choca el robot, sino poder predecir con cierta anterioridad cuando se va a producir un choque para de este modo conseguir que el robot reaccione antes de chocar y pueda de esta manera evitarlo.

Para realizar el preprocesado se hizo un programa Java, el cual consistía en extraer del fichero que almacenaba los datos obtenidos del simulador los distintos registros que almacenaba. Una vez que se obtienen todos los registros, se valida por cada uno de ellos si está o no chocando, en el caso de que no esté chocando se comprobará en que instante posterior chocará. De este modo se obtiene una nueva colección de registros en los cuales se podrá determinar si en ciertos instantes de tiempo habrá choque o no; la distancia en tiempo para esa "predicción" del choque será un valor pasado por parámetro.

Adicionalmente, en este preprocesado de datos, se generó un fichero de salida con el formato necesario para poder introducirlo en la aplicación *Weka* y poder trabajar estos datos y aplicar con ellos minería de datos.

Para que quede algo más claro el preprocesado de estos datos, se puede ver la figura 4.29 y 4.30, las cuales muestra unos datos de entrada y la correspondiente salida de datos preprocesados.

```
                                                ::::::::
                                                ::::::::
                                                ::::::::
>>lectura sonar: cero: 4944 uno: 5000 dos: 5000 tres: 5000 cuatro: 633 cinco: 741 seis: 5000 siete: 4844
velocidad: 160  no_choca: 0 segundos: 10
>>lectura sonar: cero: 3724 uno: 5000 dos: 5000 tres: 5000 cuatro: 503 cinco: 593 seis: 5000 siete: 4844
velocidad: 119  no_choca: 0 segundos: 11
>>lectura sonar: cero: 3724 uno: 5000 dos: 5000 tres: 5000 cuatro: 408 cinco: 485 seis: 5000 siete: 4844
velocidad: 92  no_choca: 0 segundos: 12
>>lectura sonar: cero: 3724 uno: 5000 dos: 5000 tres: 5000 cuatro: 326 cinco: 391 seis: 4998 siete: 4844
velocidad: 69  no_choca: 0 segundos: 13
>>lectura sonar: cero: 3724 uno: 5000 dos: 5000 tres: 265 cuatro: 265 cinco: 322 seis: 487 siete: 4844
velocidad: 52  no_choca: 0 segundos: 14
>>lectura sonar: cero: 3724 uno: 5000 dos: 5000 tres: 230 cuatro: 230 cinco: 282 seis: 433 siete: 4844
velocidad: 43  choca: 1 segundos: 15
>>lectura sonar: cero: 5000 uno: 3876 dos: 5000 tres: 5000 cuatro: 5000 cinco: 212 seis: 254 siete: 5000
velocidad: 0  no_choca: 0 segundos: 16
>>lectura sonar: cero: 5000 uno: 5000 dos: 5000 tres: 5000 cuatro: 3701 cinco: 5000 seis: 5000 siete:
5000  velocidad: 39  no_choca: 0 segundos: 17
>>lectura sonar: cero: 3732 uno: 5000 dos: 5000 tres: 5000 cuatro: 5000 cinco: 4782 seis: 3640 siete:
5000  velocidad: 1000  no_choca: 0 segundos: 18
>>lectura sonar: cero: 4140 uno: 5000 dos: 5000 tres: 5000 cuatro: 5000 cinco: 4686 seis: 4402 siete:
5000  velocidad: 921  no_choca: 0 segundos: 19
>>lectura sonar: cero: 3754 uno: 5000 dos: 5000 tres: 5000 cuatro: 5000 cinco: 4429 seis: 4046 siete:
3505  velocidad: 1000  no_choca: 0 segundos: 20
>>lectura sonar: cero: 2952 uno: 5000 dos: 5000 tres: 5000 cuatro: 5000 cinco: 4066 seis: 3666 siete:
2950  velocidad: 851  no_choca: 0 segundos: 21
>>lectura sonar: cero: 5000 uno: 5000 dos: 5000 tres: 5000 cuatro: 4761 cinco: 3624 seis: 3247 siete:
3114  velocidad: 764  no_choca: 0 segundos: 22
>>lectura sonar: cero: 5000 uno: 5000 dos: 5000 tres: 5000 cuatro: 4239 cinco: 3213 seis: 2872 siete:
3419  velocidad: 805  no_choca: 0 segundos: 23
>>lectura sonar: cero: 5000 uno: 5000 dos: 5000 tres: 5000 cuatro: 3714 cinco: 2807 seis: 2505 siete:
2979  velocidad: 694  no_choca: 0 segundos: 24
                                                ::::::::
                                                ::::::::
                                                ::::::::
```

Figura 4.29: Ejemplo de datos recogidos

```
@relation datosRobot

@attribute sonar_cero real
@attribute sonar_uno real
@attribute sonar_dos real
@attribute sonar_tres real
@attribute sonar_cuatro real
@attribute sonar_cinco real
@attribute sonar_seis real
@attribute sonar_siete real
@attribute velocidad real
@attribute choca {SI, NO}
@attribute segundos real
@attribute tiempo_choque real
@attribute chocara {SI, NO}
@data

                                    ::::::::
                                    ::::::::
                                    ::::::::
4944, 5000, 5000, 5000, 633, 741, 5000, 4844, 160, NO, 10, 5, NO
3724, 5000, 5000, 5000, 503, 593, 5000, 4844, 119, NO, 11, 4, NO
3724, 5000, 5000, 5000, 408, 485, 5000, 4844, 92, NO, 12, 3, SI
3724, 5000, 5000, 5000, 326, 391, 4998, 4844, 69, NO, 13, 2, SI
3724, 5000, 5000, 265, 265, 322, 487, 4844, 52, NO, 14, 1, SI
3724, 5000, 5000, 230, 230, 282, 433, 4844, 43, SI, 15, 0, SI
5000, 3876, 5000, 5000, 5000, 212, 254, 5000, 0, NO, 16, 27, NO
5000, 5000, 5000, 5000, 3701, 5000, 5000, 5000, 39, NO, 17, 26, NO
3732, 5000, 5000, 5000, 5000, 4782, 3640, 5000, 1000, NO, 18, 25, NO
4140, 5000, 5000, 5000, 5000, 4686, 4402, 5000, 921, NO, 19, 24, NO
3754, 5000, 5000, 5000, 5000, 4429, 4046, 3505, 1000, NO, 20, 23, NO
2952, 5000, 5000, 5000, 5000, 4066, 3666, 2950, 851, NO, 21, 22, NO
5000, 5000, 5000, 5000, 4761, 3624, 3247, 3114, 764, NO, 22, 21, NO
5000, 5000, 5000, 5000, 4239, 3213, 2872, 3419, 805, NO, 23, 20, NO
5000, 5000, 5000, 5000, 3714, 2807, 2505, 2979, 694, NO, 24, 19, NO
                                    ::::::::
                                    ::::::::
                                    ::::::::
```

Figura 4.30: Ejemplo de datos procesados

Este fichero que contiene los datos preprocesados, se carga en la aplicación "*Weka"*. En este programa, se utilizaran los atributos del fichero, a excepción de las variables "choca", "segundos" y "tiempo_choque", ya que estas variables no aportan información relevante a utilizar por un clasificador. Es reseñable el caso de la variable "*velocidad*", ya que se han realizado pruebas tanto con ella como sin ella.

La técnica de aprendizaje automático utilizada es clasificación. Por tanto, el objetivo en el uso de *"Weka"* es el de obtener unas reglas o árbol de decisión para los casos en los que se producen choque y los que no se producen, dependiendo de las lecturas de los sensores del robot, y en el caso que corresponda, de la velocidad del mismo.

El algoritmo utilizado para la generación del árbol de decisión ha sido C4.5 (J48 de Weka). Se seleccionó este algoritmo ya que, de los algoritmos que permiten utilizar variables numéricas (lectura de los sónares) para decidir la clasificación y tener una variable nominal a clasificar (si chocaría o no el robot, es decir la variable "*chocara*" que se ha visto en ejemplos anteriores), éste es el algoritmo cuya clasificación y representación de la misma es más manejable y comprensible.

El mejor resultado efectivo para la detección de obstáculos que se obtiene una vez que se prueba en simulador y robot real, fue el obtenido al aplicar el ya mencionado algoritmo C4.5 sin tener en cuenta la variable velocidad y con una predicción de choque de tres segundos, es decir que al preprocesar se indicará el choque en una situación 3 segundos antes de que se produzca, para con ello conseguir que el robot tenga tiempo de evitar el choque antes de que se produzca. El árbol de decisión que se genera tiene las siguientes características:

```
Número de hojas del árbol: 138

Tamaño del árbol: 275 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 97.2583%

 === Matriz de confusión ===

     a     b   <-- clasificado como
  6730   509 |     a = SI CHOCA
   408 25800 |     b = NO CHOCA
```

Al observar el tamaño del árbol de clasificación se aprecia que es extremadamente grande y ya que dicho árbol es necesario pasarlo a formato interpretable por C++ (lenguaje de programación en el que se está desarrollando la arquitectura) se ha desarrollado un conversor en JAVA de este árbol en formato Weka a una estructura If-Then-Else para C++.

- ***Experimentación***

Hasta llegar al árbol final que se ha mostrado anteriormente se han tenido que realizar pruebas con distintos parámetros y procesamientos, lo que conllevaba distintos árboles de clasificación. A continuación se comentará la experimentación realizada.

El primer árbol que se generó fue obteniendo en el instante de choque en el simulador las lecturas obtenidas de los ocho sónares del robot y teniendo en cuenta la velocidad a la hora de obtener el árbol. Por otro lado, el preprocesado de datos se realizó de tal modo que se determinaba la predicción de choque dos instantes de tiempo antes de que éste se produjera, es decir si por ejemplo se produce en el instante de tiempo 15, el preprocesado indicará en el instante 13 que se producirá dicho choque. Los resultados del árbol obtenido se muestran en la tabla 5.

```
Número de hojas del árbol: 142

Tamaño del árbol: 283 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 98.3253%

 === Matriz de confusión ===

     a     b   <-- clasificado como
  3212   329 |     a = SI CHOCA
   284 31605 |     b = NO CHOCA
```

Tabla 5: Árbol de clasificación inicial

Al probar este árbol en el simulador, el robot no conseguía moverse de manera correcta, en función de la velocidad el robot giraba intentando evitar obstáculos que en muchos casos no existían. El motivo de dicho problema es que en la generación del árbol el valor de la velocidad era determinante, en muchas ocasiones, a la hora de determinar el choque; este hecho provocaba que un parámetro cuyo valor es tan variable a la hora del movimiento del robot, provocaba situaciones engañosas para el mismo, por ejemplo para velocidades menores a 1.7 m/s el árbol determinaba que había choque, cuando en realidad esta velocidad podría ser establecida por defecto y no quería decir que habría choque.

Este árbol no llegó a evaluarse en el entorno real, ya que al no funcionar correctamente en el simulador, con un entorno más controlado ante la incertidumbre, ruido e imprevistos, no funcionaría tampoco correctamente en un medio real.

Con los mismos datos obtenidos para el árbol anterior, se obtuvo uno nuevo pero esta vez sin tener en cuenta la variable velocidad a la hora de generar el árbol. Los resultados del árbol de decisión que se obtuvieron se muestran en la tabla 6.

```
Número de hojas del árbol: 118

Tamaño del árbol: 235 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 98.3225%

 === Matriz de confusión ===

     a     b   <-- clasificado como
  3196   345 |     a = SI CHOCA
   249 31620 |     b = NO CHOCA
```

Tabla 6: Árbol de clasificación con variable velocidad

Como se puede apreciar, el tamaño de este nuevo árbol es muy similar al obtenido anteriormente. El porcentaje de acierto en la clasificación también se mantiene prácticamente igual.

Al probar este árbol en el simulador se obtienen los resultados deseados. Se probó este árbol de decisión en los diferentes entornos en los que se obtuvieron los datos (mostrados anteriormente).

En estas pruebas se dieron 240 situaciones de choque, de las cuales tan solo en 4 situaciones se produjo un choque, esto supone un 1.66% de choques, valor que se puede considerar aceptable.

Este mismo árbol se ejecutó en el robot físico y en un entorno no preparado. Ante esta nueva situación el robot no aporta los resultados que daba en el simulador. En esta nueva situación, el robot detecta tarde el obstáculo, y cuando intenta girar para evitarlo choca con él. Esto se debe a que en el simulador no se tiene en cuenta el acople de los parachoques que posee el robot físico, acople que aumenta la planta del robot, con lo cual ese aumento de planta es lo que provoca el choque.

Ante este nuevo problema, se generaron nuevos datos también en el simulador pero aportando este tamaño mayor de la planta del robot, el cual era de 10cm.

Se generó un nuevo árbol de decisión aplicando el algoritmo C4.5, teniendo como parámetros la lectura de todos los sónares que posee el robot, un total de ocho, y habiendo realizado un preprocesado de datos con predicción de choque a dos instantes de tiempo, tal y como se había realizado hasta ahora.

El árbol que se obtuvo tenía las características mostradas en la tabla 7.

```
Número de hojas del árbol: 124

Tamaño del árbol: 247 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 97.3779%

 === Matriz de confusión ===

     a     b   <-- clasificado como
  5200   465 |     a = SI CHOCA
   412 27370 |     b = NO CHOCA
```

Tabla 7: Árbol de clasificación con aumento de planta del robot

En este árbol se pierde algo de precisión en la clasificación de las instancias, pero aun así sigue siendo alta.

Pasado este árbol al comportamiento y siendo ejecutado en el simulador, y tal y como sucedía con el anterior árbol los resultados son buenos, hecho que cabía esperarse ya que tan sólo se ha aumentado el tamaño del robot, con lo cual determinará antes el choque que antes pero lo determinará.

Los resultados obtenidos fueron: de 380 situaciones en las que el robot chocaría si seguía moviéndose, producen 5 choques dando con ello un 1.31% en los choques, o lo que es lo mismo un 98,69% de situaciones salvadas correctamente.

Con estos resultados que se consideraron aceptables, se pasó a probar este árbol en un entorno real. Los resultados obtenidos en esta ocasión fueron bastante mejores que los anteriores, pero aún así no los deseados, ya que en un número elevados de circunstancias el robot intenta girar al detectar el obstáculo, pero la parte trasera del robot roza el obstáculo.

En primera instancia se pensó que el problema podría venir provocado que los datos se obtuvieran del simulador, por la falta de ruido en los datos, mientras que en un entorno real, los datos que obtienen los sensores sí que tienen ruido. Para intentar paliar esta situación se obtuvieron nuevos datos pero esta vez simulando el ruido en el mismo algoritmo de obtención de datos; para simular dicho ruido se introdujeron pequeñas variaciones aleatorias a las lecturas de los sensores. Con estos datos se generó un nuevo árbol con las mismas condiciones que anteriormente.

El árbol que se obtuvo poseía las características mostradas en la tabla 8.

```
Número de hojas del árbol: 296

Tamaño del árbol: 591 niveles

Tipo de validación: Cruzada

% de instancias clasificadas correctamente: 96.6624%

 === Matriz de confusión ===

      a     b   <-- clasificado como
   5943   673 |     a = SI CHOCA
    507 28232 |     b = NO CHOCA
```

Tabla 8: Árbol de clasificación con la inclusión de ruido en los datos

Al probar este nuevo árbol tanto en el simulador como en un entorno real, los resultados son muy similares al caso anterior, no solventándose el problema encontrado.

Tras esta prueba se volvió a los datos sin inclusión de ruido, pero se realizó un preprocesado con una predicción de tres instantes de tiempo, lo que llevó al árbol de clasificación final ya comentado.

4.4. Página WEB

Se ha realizado una página Web en la que se ha recopilado toda la información del presente proyecto fin de carrera, "*Una Arquitectura de Subsunción para robots Pioneer P3-DX*".

En esta página web se puede encontrar:

- Información sobre el modelo de robots de movimientos utilizados, Pioneer P3-DX.
- Información sobre la arquitectura de subsunción.
- Arquitectura desarrollada.
- Videos demostración de los resultados.

- Manual de usuario para la interacción con el robot.
- API para robots Pioneer P3-DX.
- Instalador de las aplicaciones MobileSim y Mapper3 (Basic).
- Código fuente de programas Java adicionales utilizados para el preprocesado de datos y conversor de árboles de decisión de Weka.

En la figura 4.31 se muestra la apariencia que posee la página Web en su página inicial.

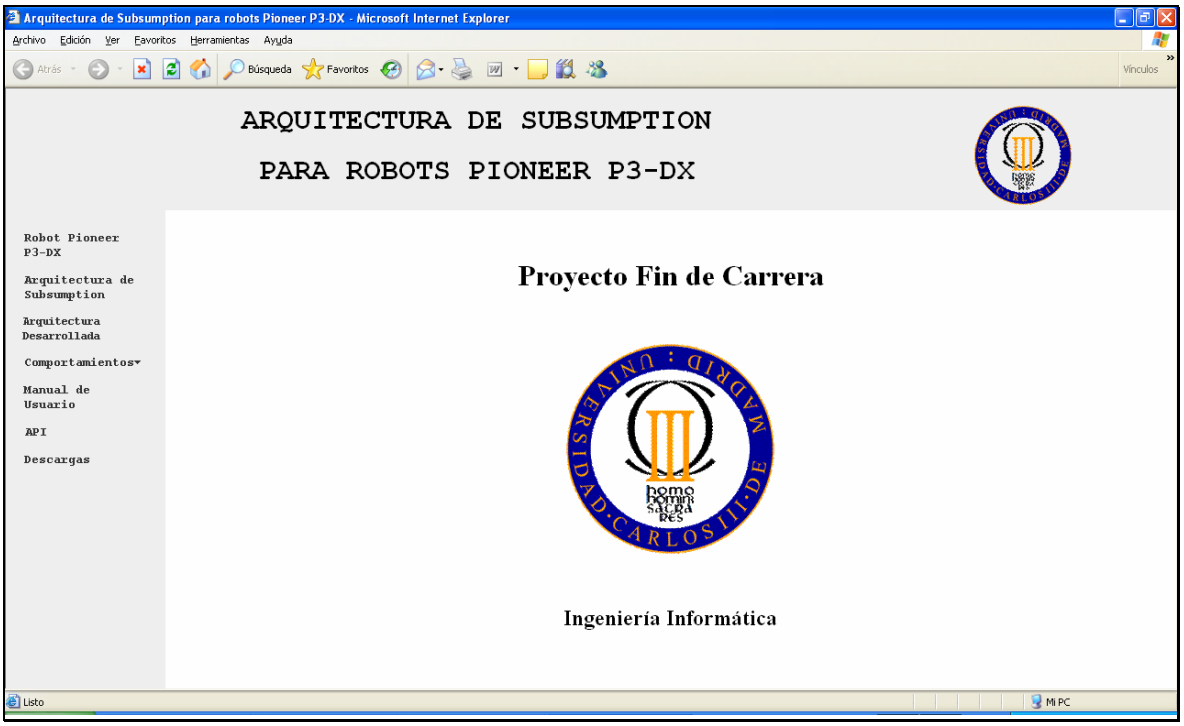

Figura 4.31: Página de Inicio de la Web

Como se ha podido observar en la figura 4.31 la Web presenta tres partes independientes:

- En su parte superior el título del presente proyecto al cual se encuentra asociada.
- Para navegar en los contenidos de la Web, se encuentra en el lateral izquierdo de la misma un menú de navegación, menú que se mantendrá siempre visible.
- En el centro de la Web se encuentra la sección en la cual se mostrarán los diferentes contenidos.

CAPÍTULO 5: **CONCLUSIONES Y TRABAJOS FUTUROS**

Una vez terminado el presente proyecto fin de carrera, se han podido extraer una serie de conclusiones, las cuales se exponen en el presente capítulo.

En este proyecto se ha trabajado con la arquitectura de subsunción, observándose una serie de ventajas en la utilización de dicha arquitectura:

- La arquitectura dota de una elevada independencia entre los distintos niveles y comportamientos de la misma. Con esta independencia se consigue que el hecho de tener que modificar uno de ellos no afecte en gran medida al resto.

- Facilidad a la hora de incluir nuevas funcionalidades en el robot. Tan solo es necesario introducir un nuevo nivel en la arquitectura, así como los comportamientos asociados a dicha funcionalidad, sin tener que alterar los ya existentes.

- El hecho de utilizar comportamientos reactivos no impide dotar al robot de ciertas funcionalidades de alto nivel. Esto es posible gracias a la posibilidad de incluir diferentes comportamientos concurrentes y coordinados mediante mecanismos de subsunción e inhibición

Adicionalmente a las ventajas ya mencionadas, con la arquitectura de subsunción se consigue un robot que se adapta mucho mejor a diversos entornos, ya que posee una parte que se puede llamar reactiva. En estas actividades reactivas se consigue que el robot consiga evitar lo que a él “le hace daño”, es decir evitar malas circunstancias para él, en el caso de nuestro robot de movimiento sería evitar el chocar con un obstáculo, pero esta arquitectura podría utilizarse para otro modelos de robots tales como aquellos

que imitan comportamientos animales para dotarles de las características primarias de aquellos animales a los que simulan.

Respecto al robot que hemos utilizado para realizar las pruebas en un medio físico, el modelo P3-DX de la familia de los robots de movimiento de Pioneer, se puede comentar que sería necesario incorporar nuevos sensores que permitan enriquecer el universo de información que percibe el robot. En el presente proyecto, para conocer el entorno que rodeaba al robot tan solo disponíamos de sensores sónares distribuidos en el diámetro del robot; adicionalmente posee parachoques tanto delanteros como traseros pero éstos sensores no permiten conocer el medio hasta que el robot choca con los objetos, con lo cual cuando un parachoques salta o es activado, quiere decir que el robot ha chocado cosa que se intentaba evitar en el mayor grado posible.

No obstante la incorporación de nuevos sensores con respecto a la arquitectura presenta gran facilidad, ya que la utilización de esos nuevos sensores en comportamientos ya desarrollados no sería estrictamente necesaria, sino que se podrían incluir nuevos comportamientos que los utilizaran y reforzaran la consistencia que ya presenta el robot en determinadas situaciones desarrolladas hasta el momento.

Por otro lado, son destacables las grandes diferencias que se han encontrado a la hora de ejecutar los comportamientos en el simulador y en el entorno real. Con esto podemos ver que las dificultades a la hora de poner en funcionamiento un comportamiento son de alguna manera imprevisibles, ya sea por que son intrínsecas a la tecnología utilizada, o por el medio cambiante en el que se mueve.

Si se ven los objetivos marcados en el capítulo 3 del presente documento, y una vez comentado el trabajo realizado, podemos concluir que se ha alcanzado el objetivo general que se marcó, crear una arquitectura de subsunción para la navegación de los robots Pioneer P3-DX. Adicionalmente, se ha experimentado con la utilización de técnicas de aprendizaje automático en el uso de dicha arquitectura, de cuyo caso se puede comentar la dificultad que supone conseguir que un robot, con una arquitectura reactiva como la desarrollada, aprender de su propia experiencia.

El presente trabajo puede ser utilizado para futuros trabajos. Una posible línea futura de trabajo nueva podría ser el dotar de comportamientos de nivel superior que permitan al robot dar una funcionalidad más completa, un ejemplo de estas nuevas tareas a realizar puede ser el mapeado de un entorno.

Otras posibles líneas de mejoras podrían venir dadas por la introducción de nuevos sensores en el robot, tal y como pudiera ser una cámara, trabajando en tal caso con el tratamiento y procesado de la imagen recibida, ya que existe una gran necesidad de investigación en este campo debido a que el procesado de las imágenes supone hasta el momento un gran coste computacional, provocando con ello un coste también elevado en tiempo; y la actuación por parte del robot ante dichas imágenes recibidas y procesadas.

Por otro lado, se podrían aplicar diferentes técnicas para mejorar el funcionamiento de los comportamientos realizados. Sería posible aplicar técnicas de aprendizaje automático para mejorar las condiciones de los algoritmos aplicados en los distintos comportamientos con respecto a las lecturas de los sensores.

BIBLIOGRAFÍA

[1] Barrientos, L.F. Peñin, C.Balaguer y R.Aracil, Fundamentos de Robótica, McGraw-Hill/Interamericana de España, S.A., Madrid 1997

[2] Brook's, R.A., A robust layered control system for a mobile robot, IEEE Journal of Robotics and Automation, RA-2, 1986, pp 14-23.

[3] Brooks, R. A., & Connell, 1. H. (1986). Asynchronous Distributed Control System for a Mobile Robot. SPIE Vol. 727 Mobile Robots, 77-84.

[4] Brooks, R.A., Elephants don't play chess, Robotics and Autonomous Systems 6, 1990, pp 3-15.

[5] Brooks, R.A., Intelligence without reason, Proc. 1991 International Joint Conference on Artificial Intelligence, 1991, pp 569-595

[6] Brooks, R.A. The behaviour language: user's guide, MIT Artificial Intelligence Laboratory Memo 1227, 1990

[7] Brooks, R. A., Connell, 1. H., & Ning, P. (1988). Herbert: A Second Generation Mobile Robot. Report No. MIT AIM-1016.

[8] Cudhea. P. H. (1988). Describing the Control Systems of Simple Robot Creatures. MIT S.M. thesis, June.

[9] Deitel, H. (2001): "C++ cómo programar". Prentice Hall.

[10] Horswill, 1. D., & Brooks, R. A. (1988). Situated Vision in a Dynamic World: Chasing Objects, t0 appear AAAI-88, St. Paul, MN.

[11] J. Borenstein, H. R. Everett, and L. Feng, Navigating Mobile Robots: Sensors and Techniques, A. K. Peter, Ltd., Wellesley, MA, 1999.

[12] J. H. Holland. Adaptation in Natural and Artificial System. The University of Michigan Press, 1975.

[13] J. I. Cox, Blance – An Experiment in guidance and navigation of an autonomous robot vehicle, IEEE Transactions on robotics and automation, 1991.

[14] J. L. Crowley, Navigation for an intelligent mobile robot, IEEE Journal of Robotic and Automation, 1985

[15] K. S. Fu, R. C. González, C. S. G. Lee, Robótica: control, detección, visión e inteligencia, McGraw-Hill, Madrid, 1988

[16] Moravec, H. P. (1984). Locomotion, vision and intelligence. In Brady & Paul (Eds.), Robotics Research (pp. 215-224). Cambridge, MA: MIT Press.

[17] Flanagan, C., Toal, D. and Strunz, R., Subsumption Control of a Mobile Robot, Proc. Polymodel-16, Sunderland, 1995, pp 150-158.

[18] Ferrell, C., Robust agent control of an autonomous robot with many sensors and actuators, MIT Artificial Intelligence Laboratory Technical Report 1443, 1993.

[19] IBM Press. IBM DB2 Intelligent Miner for Data: Utilización de Intelligent Miner for Data. IBM Press, USA, 2002.

[20] Jones, J. “Robot Programing; A practical guide to Behavior Based robotics”, TAB Robotic-McGrawHill,2002.

[21] Schildt, H. (1995). “C++. Guía de Autoenseñanza”. Ed. Osborne, McGraw-Hill.

[22] U. Nehmzow, H. Kuljis, R. Paul, R. Thomas, Mobile Robotics: A practical Introduction: History, Design, Analysis and Examples, Springer-Verlag, London,2000.

[23] Ian H. Witten, Eibe Frank, “Data Mining: Practical Machine Learning Tools and Techniques”, Morgan Kaufmann Publisher, 2000. Capítulo 8: Machine Learning Algorithms in Java (WEKA)

[24] George A. Bekey, “Autonomous Robots: From Biological Inspiration to Implementation and Control”, The MIT Press.

[25] Murphy, R. “Introduction to AI robotics”. Bradfor Book, 2002.

[26] Maja J Matari´c, “The Robotics Primer”, The MIT Press.

[27] http://robots.MobileRobots.com/

ANEXOS

5.1. Anexo 1: Árbol de decisión para detección de obstáculos

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     datosRobot-weka.filters.unsupervised.attribute.Remove-R9-12
Instances:    33447
Attributes:   9
              sonar_cero
              sonar_uno
              sonar_dos
              sonar_tres
              sonar_cuatro
              sonar_cinco
              sonar_seis
              sonar_siete
              chocara
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

sonar_dos <= 359
|   sonar_uno <= 439: SI (2845.0/83.0)
|   sonar_uno > 439
|   |   sonar_tres <= 1014: SI (149.0/2.0)
|   |   sonar_tres > 1014
|   |   |   sonar_tres <= 4025: NO (8.0)
|   |   |   sonar_tres > 4025: SI (15.0)
sonar_dos > 359
|   sonar_cinco <= 371
|   |   sonar_cinco <= 343
|   |   |   sonar_siete <= 272
|   |   |   |   sonar_uno <= 3080: SI (42.0/3.0)
|   |   |   |   sonar_uno > 3080
|   |   |   |   |   sonar_uno <= 4952
|   |   |   |   |   |   sonar_dos <= 2908: SI (6.0)
|   |   |   |   |   |   sonar_dos > 2908
|   |   |   |   |   |   |   sonar_dos <= 3219: NO (8.0)
|   |   |   |   |   |   |   sonar_dos > 3219
|   |   |   |   |   |   |   |   sonar_siete <= 242: NO (7.0/1.0)
|   |   |   |   |   |   |   |   sonar_siete > 242: SI (25.0/7.0)
|   |   |   |   |   sonar_uno > 4952: SI (82.0/15.0)
|   |   |   sonar_siete > 272
|   |   |   |   sonar_cinco <= 316: SI (2006.0/2.0)
|   |   |   |   sonar_cinco > 316
|   |   |   |   |   sonar_uno <= 1325
|   |   |   |   |   |   sonar_dos <= 830: SI (36.0)
|   |   |   |   |   |   sonar_dos > 830
|   |   |   |   |   |   |   sonar_uno <= 914: NO (7.0)
|   |   |   |   |   |   |   sonar_uno > 914: SI (9.0/1.0)
|   |   |   |   |   sonar_uno > 1325
|   |   |   |   |   |   sonar_siete <= 4067: SI (329.0/1.0)
|   |   |   |   |   |   sonar_siete > 4067
|   |   |   |   |   |   |   sonar_siete <= 4560: NO (5.0)
|   |   |   |   |   |   |   sonar_siete > 4560: SI (22.0)
|   |   sonar_cinco > 343
|   |   |   sonar_cuatro <= 416
|   |   |   |   sonar_siete <= 658
|   |   |   |   |   sonar_seis <= 365
|   |   |   |   |   |   sonar_siete <= 626: SI (12.0)
|   |   |   |   |   |   sonar_siete > 626
|   |   |   |   |   |   |   sonar_dos <= 4292: SI (6.0/1.0)
```

```
|   |   |   |   |   |   |   sonar_dos > 4292: NO (2.0)
|   |   |   |   |   |   sonar_seis > 365: NO (5.0)
|   |   |   |   |   sonar_siete > 658: SI (93.0/4.0)
|   |   |   |   sonar_cuatro > 416
|   |   |   |   |   sonar_cinco <= 354
|   |   |   |   |   |   sonar_tres <= 624: NO (10.0/2.0)
|   |   |   |   |   |   sonar_tres > 624: SI (102.0/14.0)
|   |   |   |   |   sonar_cinco > 354
|   |   |   |   |   |   sonar_seis <= 319
|   |   |   |   |   |   |   sonar_tres <= 2961: SI (44.0/8.0)
|   |   |   |   |   |   |   sonar_tres > 2961
|   |   |   |   |   |   |   |   sonar_tres <= 4570
|   |   |   |   |   |   |   |   |   sonar_cero <= 4636: SI (4.0/1.0)
|   |   |   |   |   |   |   |   |   sonar_cero > 4636: NO (12.0/1.0)
|   |   |   |   |   |   |   |   sonar_tres > 4570: SI (7.0)
|   |   |   |   |   |   sonar_seis > 319
|   |   |   |   |   |   |   sonar_siete <= 2297: NO (100.0/34.0)
|   |   |   |   |   |   |   sonar_siete > 2297
|   |   |   |   |   |   |   |   sonar_cero <= 4648
|   |   |   |   |   |   |   |   |   sonar_cero <= 3547
|   |   |   |   |   |   |   |   |   |   sonar_dos <= 1385: NO (2.0)
|   |   |   |   |   |   |   |   |   |   sonar_dos > 1385: SI (7.0)
|   |   |   |   |   |   |   |   |   sonar_cero > 3547: NO (3.0)
|   |   |   |   |   |   |   sonar_cero > 4648: SI (9.0)
|   sonar_cinco > 371
|   |   sonar_tres <= 436
|   |   |   sonar_tres <= 361: SI (212.0)
|   |   |   sonar_tres > 361
|   |   |   |   sonar_siete <= 4759
|   |   |   |   |   sonar_siete <= 3838
|   |   |   |   |   |   sonar_seis <= 440: NO (8.0/1.0)
|   |   |   |   |   |   sonar_seis > 440: SI (114.0/27.0)
|   |   |   |   |   sonar_siete > 3838: NO (22.0/3.0)
|   |   |   |   sonar_siete > 4759: SI (93.0/23.0)
|   |   sonar_tres > 436
|   |   |   sonar_cuatro <= 438
|   |   |   |   sonar_seis <= 1196
|   |   |   |   |   sonar_siete <= 698: NO (14.0/1.0)
|   |   |   |   |   sonar_siete > 698
|   |   |   |   |   |   sonar_cuatro <= 429: SI (37.0/10.0)
|   |   |   |   |   |   sonar_cuatro > 429
|   |   |   |   |   |   |   sonar_dos <= 1697: NO (15.0/3.0)
|   |   |   |   |   |   |   sonar_dos > 1697: SI (3.0)
|   |   |   |   sonar_seis > 1196
|   |   |   |   |   sonar_cuatro <= 411: SI (125.0)
|   |   |   |   |   sonar_cuatro > 411
|   |   |   |   |   |   sonar_uno <= 1837: NO (3.0)
|   |   |   |   |   |   sonar_uno > 1837
|   |   |   |   |   |   |   sonar_tres <= 4904: SI (23.0/1.0)
|   |   |   |   |   |   |   sonar_tres > 4904: NO (2.0)
|   |   |   sonar_cuatro > 438
|   |   |   |   sonar_seis <= 325
|   |   |   |   |   sonar_seis <= 266
|   |   |   |   |   |   sonar_siete <= 240
|   |   |   |   |   |   |   sonar_seis <= 250: NO (3.0)
|   |   |   |   |   |   |   sonar_seis > 250: SI (24.0/6.0)
|   |   |   |   |   |   sonar_siete > 240: SI (72.0)
|   |   |   |   |   sonar_seis > 266
|   |   |   |   |   |   sonar_cinco <= 390: NO (20.0/2.0)
|   |   |   |   |   |   sonar_cinco > 390
|   |   |   |   |   |   |   sonar_siete <= 243
|   |   |   |   |   |   |   |   sonar_cero <= 1144: SI (12.0/1.0)
|   |   |   |   |   |   |   |   sonar_cero > 1144: NO (104.0/49.0)
|   |   |   |   |   |   |   sonar_siete > 243: SI (44.0)
|   |   |   |   sonar_seis > 325
|   |   |   |   |   sonar_uno <= 327
|   |   |   |   |   |   sonar_uno <= 259
|   |   |   |   |   |   |   sonar_tres <= 769
```

```
|   |   |   |   |   |   |   |   sonar_dos <= 373: SI (26.0/8.0)
|   |   |   |   |   |   |   |   sonar_dos > 373: NO (4.0)
|   |   |   |   |   |   |   sonar_tres > 769: SI (116.0/2.0)
|   |   |   |   |   |   sonar_uno > 259
|   |   |   |   |   |   |   sonar_cero <= 2965
|   |   |   |   |   |   |   |   sonar_cero <= 359
|   |   |   |   |   |   |   |   |   sonar_cero <= 239: NO (136.0/55.0)
|   |   |   |   |   |   |   |   |   sonar_cero > 239
|   |   |   |   |   |   |   |   |   |   sonar_tres <= 1707
|   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 4988
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 571: NO (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 571: SI (7.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_siete > 4988
|   |   |   |   |   |   |   |   |   |   |   |   sonar_tres <= 562
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 4148: NO (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 4148
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 356: SI
(8.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 356: NO
(3.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_tres > 562: NO (13.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_tres > 1707: SI (16.0/1.0)
|   |   |   |   |   |   |   |   |   sonar_cero > 359
|   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 1107
|   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 1508: SI (5.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_siete > 1508
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 3372
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 324: NO (19.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno > 324: SI (2.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_siete > 3372
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 3936: SI (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 3936
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 4418: SI (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 4418: NO
(14.0/4.0)
|   |   |   |   |   |   |   |   |   |   sonar_cuatro > 1107: NO (88.0/3.0)
|   |   |   |   |   |   |   |   sonar_cero > 2965
|   |   |   |   |   |   |   |   |   sonar_uno <= 314: SI (19.0)
|   |   |   |   |   |   |   |   |   sonar_uno > 314: NO (4.0/1.0)
|   |   |   |   |   sonar_uno > 327
|   |   |   |   |   |   sonar_siete <= 263
|   |   |   |   |   |   |   sonar_cero <= 452: SI (34.0/1.0)
|   |   |   |   |   |   |   sonar_cero > 452
|   |   |   |   |   |   |   |   sonar_siete <= 233
|   |   |   |   |   |   |   |   |   sonar_cinco <= 4786: SI (13.0)
|   |   |   |   |   |   |   |   |   sonar_cinco > 4786
|   |   |   |   |   |   |   |   |   |   sonar_seis <= 4392: NO (2.0)
|   |   |   |   |   |   |   |   |   |   sonar_seis > 4392: SI (4.0)
|   |   |   |   |   |   |   |   sonar_siete > 233: NO (221.0/63.0)
|   |   |   |   |   |   sonar_siete > 263
|   |   |   |   |   |   |   sonar_cero <= 294
|   |   |   |   |   |   |   |   sonar_cero <= 232: SI (41.0/2.0)
|   |   |   |   |   |   |   |   sonar_cero > 232
|   |   |   |   |   |   |   |   |   sonar_uno <= 663
|   |   |   |   |   |   |   |   |   |   sonar_cero <= 253
|   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 2907
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 3072: NO (13.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 3072: SI
(28.0/11.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_siete > 2907
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 246
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 1961
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 242: SI
(3.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 242: NO
(2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 1961
```

```
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 245: NO
(39.0/3.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 245: SI
(3.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 246: NO (25.0)
|   |   |   |   |   |   |   |   |   |   sonar_cero > 253: NO (200.0/4.0)
|   |   |   |   |   |   |   |   |   sonar_uno > 663
|   |   |   |   |   |   |   |   |   |   sonar_tres <= 4837
|   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 3554
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 3023: NO (5.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 3023
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 274
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 4948: SI
(7.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 4948
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 249: SI
(2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 249: NO
(4.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 274: NO (3.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_uno > 3554
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 3841
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos <= 3712: NO (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos > 3712: SI
(4.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 3841: NO (35.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_tres > 4837
|   |   |   |   |   |   |   |   |   |   |   |   sonar_dos <= 3952: NO (5.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_dos > 3952
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 1167
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 249: SI (3.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 249: NO (7.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 1167: SI (53.0/2.0)
|   |   |   |   |   |   |   |   sonar_cero > 294
|   |   |   |   |   |   |   |   |   sonar_dos <= 383
|   |   |   |   |   |   |   |   |   |   sonar_dos <= 366
|   |   |   |   |   |   |   |   |   |   |   sonar_dos <= 360: NO (2.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_dos > 360
|   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 3566: SI (8.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 3566
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 4472: NO (3.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 4472: SI (9.0/3.0)
|   |   |   |   |   |   |   |   |   |   sonar_dos > 366
|   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 1157
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 556: NO (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 556
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 380: SI (7.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno > 380: NO (3.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_siete > 1157: NO (88.0/11.0)
|   |   |   |   |   |   |   |   |   sonar_dos > 383
|   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 542
|   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 768
|   |   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 4154: NO (563.0/22.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_seis > 4154
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 4760: NO (46.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 4760: SI (9.0)
|   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 768
|   |   |   |   |   |   |   |   |   |   |   |   sonar_dos <= 3137
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 4268: NO (27.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 4268
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos <= 2013: SI (9.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos > 2013: NO (7.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_dos > 3137: SI (15.0)
|   |   |   |   |   |   |   |   |   |   sonar_cuatro > 542
|   |   |   |   |   |   |   |   |   |   |   sonar_seis <= 363
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 474: NO (88.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 474
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 4139: SI (11.0)
```

```
|   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 4139: NO (3.0/1.0)
|   |   |   |   |   |   |   |   |   |   sonar_seis > 363
|   |   |   |   |   |   |   |   |   |   |   sonar_tres <= 507
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 595
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 976: NO (12.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 976
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_tres <= 441: NO
(2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_tres > 441: SI
(8.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 595
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 2084: NO
(201.0/4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 2084
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 4468
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 393: SI
(3.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno > 393: NO
(49.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 4468
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 2859:
SI (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 2859
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 784:
SI (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno > 784:
NO (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_tres > 507
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 411
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <= 386: SI (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 386: NO (22.0)
|   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco > 411
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 478
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete <= 1076
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro <= 4066:
NO (89.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cuatro > 4066
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco <=
4278: NO (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cinco >
4278
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete
<= 437: NO (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete >
437
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos
<= 4143: NO (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_dos
> 4143: SI (21.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_siete > 1076
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero <= 341
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <= 613
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno <=
534: SI (6.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno >
534: NO (6.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_uno > 613:
NO (210.0/5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 341: NO
(847.0/4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   sonar_cero > 478: NO
(22822.0/68.0)

Number of Leaves  :     138

Size of the tree :      275
```

```
Time taken to build model: 10.8 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       32530               97.2583 %
Incorrectly Classified Instances       917                2.7417 %
Kappa statistic                          0.9188
Mean absolute error                      0.037
Root mean squared error                  0.1495
Relative absolute error                 10.9012 %
Root relative squared error             36.2963 %
Total Number of Instances            33447

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.93      0.016      0.943     0.93      0.936    SI
  0.984     0.07       0.981     0.984     0.983    NO

=== Confusion Matrix ===

     a     b   <-- classified as
  6730   509 |     a = SI
   408 25800 |     b = NO
```

5.2. Anexo 2: Especificaciones para Robots P3-DX

Características Físicas	
longitud (cm)	44.5
Ancho (cm)	39.3
Alto (cm)	23.7
Altura sobre el suelo (cm)	6.0
Altura sobre el suelo con parachoques (cm)	3.5
Peso (kg)	9

Energía	
Baterías 12VDC	3
Tiempo de ejecución (horas)	8-10
Tiempo de recarga (hr/batería)	6

Movilidad	
ruedas	2
diámetro (mm)	195.3
anchura (mm)	47.4
Tracción	diferencial
Max vel traslación (mm/seg)	1.400
Max vel rotación (mm/seg)	300

Sensores	
Sónares delanteros	8
Sónares traseros	8 (opcional)
Encoders (cuenteo/rev)	76,6
Parachoques	opcional

Control y Puertos	
Fuente de energía	estándar
carga	estándar
Joystick	opcional
parada de motor	opcional
Fuente auxiliar 1	5 & 12 VDC sw'd
Fuente auxiliar 2	5 & 12 VDC sw'd
Sistema serial	estándar
motores	estándar
reseteo de micro controles	estándar

5.3. Anexo 3: Manual de simulador MobileSim

- ***Instalación***

Para instalar MobileSim en cualquiera de los paquetes de las RPM o Debian (den) debes usar las herramientas de instalaciones apropiadas al sistema (`rpm` en RedHat, SuSe, etc y `dpkg` o Synaptic ("Añadir los paquetes descargados") en Debian, Ubuntu, etc.). El RPM fue construido en RedHat 7,3, y el paquete de Debian se basa en Debian 3.1. MobileSim requiere la GTK + 2.0 bibliotecas en RedHat, y GTK + 2.6 en Debian. También requiere libstdc++ 2.2 para libc6 (Estas bibliotecas son piezas estándar de la mayoría de las instalaciones de Linux, pero si que se están perdiendo puede que tenga que instalar los paquetes que ofrecen los mismos). Para instalar MobileSim en Windows, ejecute el auto-ejecutable de instalación.

MobileSim es un software libre que podemos encontrar en la Web http://robots.mobilerobots.com/MobileSim.

- ***Ejecución***

MobileSim puede ser ejecutado desde el menú Inicio (Windows), o la de Gnome o menú KDE (Linux), o desde línea de comandos (cualquier plataforma).

Se puede ejecutar MobileSim por línea de comando de siguiente manera:

```
MobileSim
```

o, para omitir el diálogo inicial de la especificación de un archivo de mapa y modelo de robot se pude utilizar la siguiente línea de comando:

```
MobileSim -m <map file> -r <robot model>
```

Por ejemplo:

```
MobileSim -m columbia.map -r p3dx
```

Los parámetros son opcionales. El modelo por defecto del robot, si no se especifica con –r, es "p3dx". Algunos modelos de robot adicionales que se incluyen son "p3at", "powerbot", "peoplebot", "Patrolbot-sh" y "amigo".

Para incluir en una simulación varios robots, se debe utilizar -r varias veces, nombrando a los robots después del modelo, separados por dos puntos. Un ejemplo de este hecho es el siguiente:

```
 MobileSim -m columbia.map -r p3dx:robot1 -r
p3dx:robot2 -r amigo:robot3
```

Si se está trabajando con muchos robots, puedes encontrar el inconveniente de tener que crear todos ellos al iniciar MobileSim. If you use -R instead of -r, then a robot "factory" is created instead of a persistent robot model; a new robot will be created for each client that connects, and destroyed when the client exits. Si utiliza -R en lugar de -r, entonces un nuevo robot se creará para cada cliente que se conecta, y destruido cuando el cliente sale.

Si ejecuta MobileSim sin opciones en la línea de comandos, o desde el menú de escritorio, puede seleccionar el mapa y un modelo de robot de un cuadro de diálogo. O bien, puede optar por no utilizar el mapa (pero hay que tener en cuenta que en este caso, el universo utilizable se limita a 200 x 200 metros, el área gris indica el limite del universo).

Al lanzar la simulación, aparecerá una ventana mostrando el mapa del entorno y robot o robots. El robot comenzará desde la posición de salida si estaba incluida en el mapa, o en el centro del mapa. El cuerpo del robot se dibuja sobre la base aproximada de longitud y anchura (incluyendo ruedas) del modelo seleccionado. You can pan in the window by holding down the right mouse button and dragging. La visualización del mapa puede desplazarse en la interfaz manteniendo pulsada el botón derecho del ratón y arrastrando. Se puede hacer Zoom con la rueda del ratón, o manteniendo pulsada el botón central del ratón y arrastrando hacia fuera o desde el centro del círculo que aparece.

El robot puede moverse arrastrando con el botón izquierdo del ratón y girar arrastrando con el botón derecho del ratón. Tenga en cuenta que esto va a actualizar la posición real del robot, pero *no a* su odometría - es como recoger el robot y llevarlo a una nueva ubicación.

Se puede activar la visualización de dispositivos usando el menú Ver. Sonar: si un programa cliente se ha conectado y activado el sonar, el centro del campo del sonar se establecerá como líneas de color gris. Láser: Si un programa cliente ha permitido que el telémetro láser, su área de distribución se utilizará como una luz azul sobre el terreno. Posición: se muestran respecto a los ejes la posición real del robot, su posición odométrica y la velocidad actual que posee.

Se puede capturar una imagen o una secuencia de imágenes de lo mostrado en la ventana de ejecución. La imagen o las imágenes se pueden guardar como un archivo de mapa de bits haciendo uso de la funcionalidad de exportación en el menú Archivo.

El panel de información inferior muestra los mensajes de registros y alertas sobre lo que los robots están realizando (algunos comandos se traducen en un mensaje de registro, pero no todos.). La barra de estado situada bajo el panel de información, muestra de izquierda a derecha: el tiempo total desde que comenzó a ejecutar el simulador, la proporción entre tiempo real y tiempo de simulación, el número total de suscripciones en la simulación (una por cada robot más una por cada dispositivo al que se accede), y finalmente MobileSim y su versión.

Una vez en funcionamiento, un programa puede conectar con el simulador a través de puerto TCP 8101, y utilizar el mismo protocolo que en el puerto TCP como lo hace con un verdadero robot lo largo de su puerto serie vínculo. Puede especificar un puerto alternativo con el número opción -p. Si varios robots se pidió con múltiples opciones -r, cada robot utilizará el próximo número de puerto (es decir, 8102, 8103, etc.).

Al ejecutar MobileSim por línea de comandos, se posee la opción - help para obtener una lista completa de opciones.

- ***Avanzada Uso***

Una lista completa de parámetros por línea de comando para MobileSim es la siguiente:

Uso:

```
 MobileSim [-m map ] [-r robot model ...]
[...options...]
```

o

```
 MobileSim -W stage world [...options...]
```

`--map` *map*	Mapa del archivo de carga (por ejemplo, creado con Mapper3)
`-m` *map*	Igual que el `-map` *map*.
`-- nomap`	Crear un mapa "vacío" para empezar. Igual que -map ""
`--` *modelo de* robot *[: nombre]*	Crear un robot de simulación de un modelo determinado. Puede repetirse con diferentes nombres y modelos para crear varios robots simulados. Por ejemplo: `-- robot p3dx - robot amigo: MyRobot-r custom.p: CustomBot`
`-` r *modelo [: nombre]*	Igual que `-` `robot` *modelo [: nombre].*
`--robot-factory` *model*	En lugar de crear un robot del modelo dado, aceptar cualquier número de programas cliente en su puerto, la creación de un nuevo ejemplo del modelo para cada cliente, y destruirlo cuando el cliente se desconecta.
`-` R *modelo*	Igual que `-` `robot de fábrica` *modelo*
`-` p *puerto*	Emula conexiones TCP indicando el puerto(por

	defecto: 8101)
`-W` *`worldfile`*	Use la instancia worldfile *worldfile* de un mapa.
`-- stageworld`	Lo mismo que-W.
`--fullscreen-gui`	Mostrar ventana en modo pantalla completa.
`--maximize-gui`	Mostrar ventana maximizada.
`--minimize-gui`	Mostrar ventana minimizada
`-- noninteractive`	No mostrar ningún cuadro de diálogo interactivo que pueda bloquear la ejecución del programa.
`--lite-graphics`	Desactivar algunos gráficos para un mejor rendimiento
`--no-graphics`	Desactivar todos los gráficos para dibujar y así obtener un ligero aumento del rendimiento
`--html`	Imprimir mensajes de registro y otros de salida en HTML en lugar de texto plano
`--cwd` *`dir`*	Cambia al directorio *dir* de inicio. Los programas clientes pueden cargar los mapas relativos a este directorio.
`--log-file` *`archivo`*	Imprimir mensajes de registro al *archivo* en lugar de error estándar consola.
`- l` *`archivo`*	Igual que - log-file *archivo.*
`--update-interval` *`ms`*	El tiempo entre cada actualización de simulación. (El valor por defecto es 100 ms).
`--update-sim-time` *`ms`*	Cantidad de tiempo de simulación para cada actualización (por defecto es igual a update-intenval. Un valor mayor que update-intenval daría los resultados con mayor rapidez que en la simulación en tiempo real, lo que puede causar problemas para los clientes)
`--start` *`x , y , th`* O `--start outside` O `--start random`	Utilice *x, y,* orientación como punto de partida del robot (incluso si el mapa posee puntos de partida). O bien, utilice la palabra clave "outside" para que el robot comience 2m fuera de los limites del mapa, o bien, utilizar la palabra clave "random" para elegir al azar un punto de partida dentro de los límites del entorno.
`--resolution` *`r`*	Utilice la resolución *r* (milímetros) para las colisiones y los sensores. El valor por defecto es 20 mm (2 cm)
`--ignore-command` *num*	Ignorar el comando, cuyo número se da.

5.4. Anexo 4: Integración Pioneer P3-DX y MobileSim

A la hora de programar un comportamiento en un robot físico P3-DX o en el simulador *MobileSim* hay que tener en cuenta el tipo de conexión que debe realizarse en cada caso.

Conexión con el simulador *MobileSim*

Si se pretende lanzar la ejecución de un comportamiento en el simulador *MobileSim* hay que asociar en modo Cliente-Sevidor al programa que contiene el comportamiento y al simulador. Para realizar esa conexión, y al tratarse del simulador ésta debe realizarse siguiendo el protocolo TCP/IP. Al establecerse una conexión TCP, se realiza una comunicación mediante socket, por lo cual es necesario indicar el puerto por el que se realizará la comunicación, así como la máquina con la que comunicarse. Por defecto, y salvo que se indique lo contrario, estos parámetros de máquina y puerto tomarán como *host* la máquina local y el puerto el utilizado por defecto por el simulador, el 8101. Una vez hecho esto ya se tiene asociado el cliente y el servidor, tan solo es necesario abrir la conexión.

Una vez que la conexión se ha realizado correctamente, se debe añadir al objeto del robot la instancia de la conexión con la que se comunicará.

Conexión con el robot

El proceso de conexión con el robot es muy similar, pero con la diferencia que ahora no se realiza la conexión por TCP, sino que se realiza una conexión serie, es decir se conecta por un puerto serie al robot.

Al igual que sucedía para la conexión TCP, hay que indicar el puerto por el que establecer y en el caso de ni indicarse se establecerá por defecto el puerto serie “COM1”.

Una vez realizado esto, se deberá abrir el puerto, estableciéndose de este modo la conexión con el robot.

Al igual que con la conexión con el simulador, una vez que la conexión se a realizado, se debe añadir al objeto del robot la instancia de la conexión con la que se comunicará.

Cabe destacar que, tal y como se ha comentado, la conexión se realiza por el puerto serie, por lo que en el caso de que el cliente no lo posea, se deberá utilizar un conversor de puerto usb en serie, creándose un puerto serie virtual. Por esto se recomienda antes de intentar establecer la conexión confirmar el nombre del puerto virtual que se crea.

Para poder ver el nombre de este puerto creado debe seguir los siguientes pasos:

1. Pulsar con el botón derecho del ratón sobre el icono de **MiPC**.
2. Seleccionar **Propiedades**.
3. Una vez dentro, ir a la pestaña **Hardware**.
4. Seleccionar **Administrador de dispositivos**.
5. Dentro de la sección **Otros dispositivos** aparecerá el nombre del puerto serie virtual.

Una vez que se establece la conexión, ya sea con el simulador o con el robot, hay que bloquearla para que de este modo pueda realizarse la comunicación propiamente dicha, ya que se disponen ambos dispositivos, cliente y servidor, a enviar y recibir paquetes de datos.

Una vez creada la conexión ya sea TCP (para simulador) o serie (para robot) se podrán añadir comportamientos a la instancia del robot que se posea.

Estos comportamientos los crea el propio usuario a partir de la herencia de una genérica. Para la realización de dichos comportamientos, el usuario puede hacer uso de la información de los sensores de los que dispone el robot.

Se pueden asociar más de un comportamiento a la instancia del robot, en este caso se encargará el sistema “revolvedor” de decidir que acción se debe ejecutar de entre las dadas por los comportamientos. Este sistema “revolvedor” se encarga de decidir que acción deberá ejecutarse dentro del abanico de posibilidades que existen, pudiendo seguir diferentes criterios a la hora de tomar la decisión; El usuario puede crear uno específico para ajustarlo al criterio de decisión que dicho usuario desee, pero por defecto el criterio de elección es la prioridad de los comportamientos, siendo más prioritario aquel comportamiento con valor más alto.

5.5. Anexo 5: Conversor de Árbol de Weka a estructura If-Then-Else

Para realizar el conversor de árbol Weka a una estructura de If-Then-Else, se ha aprovechado el hecho de que en el árbol utiliza el carácter ‘ | ‘ para indicar la profundidad, y que todo nivel tiene asociado la condición afirmativa de una variable (*If*) y la negativa (*else*). Se puede observar esta situación en el siguiente ejemplo:

```
sonar_uno <= 1325
|   sonar_dos <= 830: SI (36.0)
|   sonar_dos > 830
|   |   velocidad <= 62: SI (7.0)
|   |   velocidad > 62: NO (9.0/1.0)
sonar_uno > 1325
|   sonar_siete <= 4067: SI (329.0/1.0)
|   sonar_siete > 4067
|   |   sonar_siete <= 4560: NO (5.0)
|   |   sonar_siete > 4560: SI (22.0)
```

El pseudocódigo del conversor realizado es el siguiente:

```
Conversor(){
      Leer línea
      Obtener nivel
      //Zona IF del árbol
      Si condición y clasificación directa
            Escribir condición y clasificación
      Si no //solo condición
            Escribir condición del IF
            Conversor() //llamada recursiva

      //Zona ELSE del árbol
      Escribir condición del Else
      Leer línea
      Obtener nivel
      Si condición y clasificación directa
            Escribir condición y clasificación
      Si no //solo condición
            Pintar condición del IF
            Conversor() //llamada recursiva
}
```

NOTA: El árbol weka se lee de fichero.

Como puede apreciarse en el pseudocódigo, el algoritmo seguido es recursivo, para conseguir introducir todos los niveles y limitar correctamente el ámbito de cada condición.